Aspects of learning within networks of spiking neurons

Carnell, Andrew

Award date:
2008

Awarding institution:
University of Bath

Aspects of Learning within Networks of Spiking Neurons.

submitted by

Andrew Robert Carnell

for the degree of Doctor of Philosophy

University of Bath

Department of Computer Science

November 2008

Abstract

Spiking neural networks have, in recent years, become a popular tool for investigating the properties and computational performance of large massively connected networks of neurons. Equally as interesting is the investigation of the potential computational power of individual spiking neurons. An overview is provided of current and relevant research into the Liquid Sate Machine, biologically inspired artificial STDP learning mechanisms and the investigation of aspects of the computational power of artificial, recurrent networks of spiking neurons.

First, it is shown that, using simple structures of spiking Leaky Integrate and Fire (LIF) neurons, a network $n(P)$, can be built to perform any program P that can be performed by a general parallel programming language. Next, a form of STDP learning with normalisation is developed, referred to as $STDP + N$ learning. The effects of applying this $STDP + N$ learning within recurrently connected networks of neurons is then investigated. It is shown experimentally that, in very specific circumstances Anti-Hebbian and Hebbian STDP learning may be considered to be approximately equivalent processes.

A metric is then developed that can be used to measure the *distance* between any two spike trains. The metric is then used, along with the $STDP + N$ learning, in an experiment to examine the capacity of a single spiking neuron that receives multiple input spike trains, to simultaneously learn many temporally precise Input/Output spike train associations.

The $STDP+N$ learning is further modified for use in recurrent networks of spiking neurons, to give the $STDP + N_{Type2}$ learning methodology. An experiment is devised which demonstrates that the Type 2 method of applying learning to the synapses of a recurrent network — effectively a randomly shifting locality of learning — can enable the network to learn firing patterns that the typical application of learning is unable to learn. The resulting networks could, in theory, be used to create to simple structures discussed in the first chapter of original work.

Acknowledgements

Many thanks to my supervisor Daniel Richardson for not only giving me the opportunity of studying on a fully funded EPSRC grant (GR/S69306) but also for offering a great deal of invaluable insight and help throughout many conversations and for always maintaining a highly positive attitude towards my work. I must also thank Leslie S. Smith, Joanna J. Bryson and Alwyn Barry for their most useful suggestions and comments. Thanks to all of the members of the departmental office and to the support staff for providing helpful and professional office support throughout the duration of my time in the department. A special thanks also, to Claire Willis for giving me the opportunity to study on the M.Sc. course, through which I developed my interest in neural networks.

Thanks to all of my friends who supported me throughout my time at Bath and to anyone who helped with my dual obsessions of work and bike racing.

Thank you Tracey and Neil for, among other things, trips to Cornwall, copious amounts of food and of course, fat cat. Thanks to Shane and my brothers Peter and Richard for drinks at the pub, games on the Nintendo and supportive chats. Many thanks to Katy for all of her encouragement, love and support over so many years, you really did help. Finally, and by no means least, a heartfelt thanks to my parents and my grandmother for providing constant support of all kinds throughout the course of my Ph.D. and indeed, throughout my entire time at University without which, it is likely that none of it would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Spiking Neural Networks: Learning and Computational Performance

The research field of artificial neural networks has been an existing and expanding area of computer science for over 50 years. Current neural net research is a highly diverse subject, insofar as focus of the research can vary from modelling and investigation of the observed structures individual neurons and synapses, such as the ion-channels that relay synaptic activity, to investigating the computational properties of structures ranging from individual neurons, to complex and highly interconnected networks consisting of many hundreds, or even thousands, of neurons and with thousands or hundreds of thousands of synapses.

The goals of this research can vary from attempting to build increasingly realistic models of the biological neurons and synapses, to furthering our understanding and knowledge of how networks of neurons may learn and remember specific neural patterns, through to investigating the computational power of such networks at a fundamental level. In addition to their interest to the academic community, neural networks, in one guise or another, are currently used in a wide range of applications throughout a variety of industries. Specific uses include: financial

forecasting (Peltarion, 2008); adaptive control systems such as the Intelligent Flight Control System (IFCS) developed at NASA (Williams-Hayes, 2005); data mining (SPSS Neural Networks, 2008). Neural networks can be a highly useful tool in recognising and categorising real-world patterns, by which it is meant; patterns that may contain elements of variability, distortion or noise. For example, the task of recognising the same face viewed in different lighting conditions, or predicting the short term trajectory of a particular stock given prior information and potentially related but novel time-series data.

While usage of neural networks may be considered to be somewhat widespread, there are still many questions regarding the fundamental capabilities and characteristics of individual, and networks of, artificial neurons, that are yet to be addressed. This thesis is concerned with investigating some of these fundamental aspects. For example, a modified form of Spike Time Dependent Plasticity (STDP) learning with normalisation is formulated ($STDP + N$ learning) and used to modify the synaptic weights to spiking Leaky Integrate and Fire (LIF) neurons. A metric is also developed that allows two spike trains to be compared and provide a measure of the *distance* between them. This metric is then used as measure of how well a single spiking LIF neuron can learn precise goal spike train patterns and to investigate how the learning of multiple I/O spike train patterns varies with the number of input connections. A framework is also established in which structures of spiking neurons are built that operate in such a way and with a degree of precision that, allows the creation of certain programming constructs — as seen in the Occam programming language (Inmos Limited, 1998).

Essentially, this thesis is concerned with investigating what single neurons and recurrently connected networks of neurons can learn and how well they can learn it. The learning regime used is a variant of the biologically-inspired STDP mechanism and novel methods for the application of this STDP learning procedure are implemented in networks of neurons; with a view to improving learning according to quantitative and observable measures.

1.2 Defining the Research Field

The area of particular relevance to this dissertation is the sub-field of neural net research that involves the use of spiking neurons. The most relevant existing research, in relation to the work in this thesis, can be summarised as follows.

All experimental work in this thesis is performed in simulation, utilising artificial neurons and synapses. The detailed review of mathematical models for a range of neuronal and synaptic models that is presented in Gerstner and Kistler (2002), represents a definitive collection of just such artificial structures, ranging from the complex ion-channel models, to the more simple and less computationally intensive models such as the Leaky Integrate and Fire (LIF) neurons. The LIF model is used in the original work presented in this dissertation is also widespread in the existing research.

The Liquid State Machine (LSM), (Maass *et al*, 2002a) and similarly the Echo State Network (ESN) (Jaeger, 2001), is a highly recurrently connected spiking neural network, typically with its neurons arranged in a 3-D columnar structure and randomly connected. It is designed with the concept of the neo-cortical micro column in mind. A subset of the neurons of the LSM receive a spiking input stimulus from an input source and this firing activity spreads throughout the network according to the randomly generated synaptic weights that, generally, remain static and are unmodified by firing activity. More recent work (Hæusler and Maass, (2007)), has used differing connectivity patterns for different layers of the LSM. These different connectivity regimes were used to investigate the impact on the computational capabilities of LSMs that exhibit a laminar structure, compared to LSMs that exhibit more homogenous connectivity — synaptic weight remain static. LSM research typically involves investigation of the Memory Capacity (MC) of theses columnar LSMs, among other performance indicators.

Just as relevant as the LSM research, is the work that investigates the potential of individual neurons to learn to perform precise spike train responses to a collection of specific spike train over multiple input channels, using STDP inspired learning, as seen in Legenstein *et al* (2005), and similarly Pfister *et al* (2006). This research

was performed concurrently with parts of the precise spike work seen in chapter 6 of this thesis. Pfister *et al* focus on determining the optimal form of the STDP learning function for different scenarios involving correlated and uncorrelated input spike trains to an individual neuron. In Legenstein *et al* (2005) it is shown that individual neurons can learn complex precise spike train responses with Hebbian type learning at the input synapses.

This thesis is concerned with furthering the understanding of the computational power of spiking neurons and indeed, of recurrent networks of spiking neurons, using as a starting point, aspects of the previously mentioned research.

1.3 Original Work

The first piece of original work, chapter 4 of this thesis, establishes a framework in which networks of spiking neurons and synapses can be used to create simple neuronal structures that facilitate the creation of a network $n(P)$, where P is a program that can be implemented in any parallel programming language such that, n is able to perform P.

The motivation for applying networks of spiking neurons in the manner outlined in chapter 4, was to investigate how such neurons can be applied to a task that is very different from their standard usage. The aim of this is to expand on what it is possible to do with structures of such neurons. Networks of spiking neurons are not typically used in such a way, and the intention of the work in chapter 4 is to investigate a novel application for these neurons, one that could be used as a basis for very different form of computing than what is generally considered their normal application.

While this work does not explicitly use the STDP learning seen in subsequent chapters, it can be thought of as integrating with the work seen in chapter 7. This work was accepted and presented as a poster presentation at the 9^{th} Neural Computation and Psychology Workshop (NCPW) 2004 and accepted for publication in the journal Theoretical Computer Science, 2007 (Carnell and Richardson,

2007).

Chapter 5 contains the second piece of original work — a form of Spike Time Dependent Plasticity (STDP) learning with normalisation, $STDP + N$ learning. This is a learning mechanism that modifies the synaptic input weights to a spiking neuron based on the interaction of the timings of pre and post-synaptic spikes interpreted via an asymmetric learning function. The normalisation procedure is applied to the input weights of a neuron, and enables the weights to achieve intermediate values between the minimum and maximum possible values.

It is shown that $STDP + N$ learning, applied to the synapses of a recurrently connected spiking neural network which receives an input spike train from a spiking source neuron, produces an approximate convergence of the synaptic weights and that the network can be considered to *habituate* to the particular input spike train. An activity link vector L is defined which defines a relationship between the pre and post synaptic firing activity of an individual neuron. Given a *good* alignment of L and the input weight vector, W, of the individual neuron, it is shown that, if the synaptic weights are allowed to change sign during modification, Hebbian and Anti-Hebbian learning can be considered to be approximately equivalent processes. Part of this work was accepted as a paper to the Brain Inspired Cognitive Systems (BICS) conference, 2006, and a more complete version has been accepted for publication in Neurocomputing, 2008 (Carnell, 2008).

The motivation for the work introduced in this chapter is to devise a biologically inspired learning regime for use in networks of LIF neurons and which is to be used in the majority of the experimental work of the thesis. It was felt that it was essential to check the integrity of the learning regime. For example, to check that the learning produced stable synaptic weight values. It was considered appropriate that, the fundamentals of the learning regime be investigated. Hence, the check on applying different learning regimes one after another (Hebbian followed by Anti-Hebbian learning), as well as checking the effect of stimulus changes. Additionally, it was felt that it would be advantageous to explain, in a more mathematical sense, the process of synaptic weight changes.

Chapter 6 describes the development of a metric for weighted time series of spikes. The metric can be used to provide a continuous, quantitative measure of the *distance* between any two spike trains. It is also shown that this metric can also be used as part of a training regime in which a spiking LIF neuron is trained to produce a specific output spike train. Furthermore, the metric is used to experimentally investigate the relationship between the number of unique I/O associations a single spiking neuron can learn and the number of inputs the neuron receives. The results of this investigation appear to show that there exists a log-linear relationship between the number of inputs and the number of I/O associations that can be learned simultaneously. The metric work of this chapter was accepted and presented as a poster presentation at the European Symposium on Artificial Neural Networks (ESANN) 2005 and subsequently accepted and published in the proceedings of ESANN 2005 (Carnell and Richardson, 2005). The latter experimental work of this chapter has been submitted for publication in the journal Neural Networks.

Most research involving spiking neurons involves examining the properties of large networks of such neurons (Maass *et al* 2002a, 2002b, 2004a, 2004b; Natschläger *et al*, 2002a, 2002b; Jaeger, 2001) among many others. Less focus has been placed on investigating the capabilities of individual spiking neurons (Pfister *et al*, 2006; Toyoizumi *et al*, 2007); Legenstein *et al*, 2005). This relative lack of research means that there are likely to be many aspects concerning the computational capabilities of individual neurons that are unknown. Therefore, the motivation for performing the experimental work that forms the latter part of chapter 6 was to further the understanding of what a single neuron with a large number of input connections, such as a neo-cortical neuron, is able to learn.

In order to investigate learning within these constructs, a measure on weighted spike train similarity would be needed, in order to be able to determine what, if indeed anything at all, had been learned. This is the motivation for devising a metric that could be used to measure similarity between weighted spike trains.

Finally, in chapter 7 the framework for $STDP+N$ learning, introduced previously in this thesis, is modified further — in terms of its application to recurrently

connected networks of spiking neurons. In this framework, standard application of STDP Hebbian learning is referred to as Type 1 learning, while the modified version is referred to as Type 2 learning. More specifically, this modification, also referred to as $STDP + N_{Type2}$ learning, involves the introduction of a randomly shifting 'locality of learning', which basically means that the network is able to learn firing patterns that it was unable to learn with the standard, Type 1, application of Hebbian learning.

It is noted that the structures borne out of the use of this Type 2 learning implementation, could be used to train networks of neurons to form the simple structures first described and built in chapter 4.

Having established the integrity of the $STDP+N$ learning regime and also having investigated aspects of the computational capabilities of individual neurons with varying, and in some cases very large, numbers of input connections using this learning regime, attention was then turned to recurrently connected networks consisting of many (hundreds) neurons and more specifically the way in which the learning regime is applied with such networks.

The motivation for the work in chapter 7, is to address the method of application of Hebbian-type learning from a perspective that has not previously been used. The aim of this is to determine if there are advantages to using different methods of application of the Hebbian STDP learning regime other than the typical method of application. The typical method of application is to have the learning regime operating on all synapses of all firing neurons of a network all of the time, during a learning epoch.

1.4 Structure of Thesis

This thesis is split into two distinct sections. The first section deals with the relevant background research and can itself, be split into two sections, background work and original work.

The first background chapter highlights and describes the research material in which the Liquid State Machine (LSM) is first introduced and the in which the computational performance of the LSM is investigated on a variety of fundamental tasks.

The second background chapter describes the most recent research involved with the investigation of obtaining precise trains from spiking neurons through modification of synaptic input weights.

The second major section of this thesis presents the original work of this thesis. Firstly, chapter 4 describes how simple structures of neurons may be used to perform the functions of a parallel programming language. In chapter 5 the learning mechanism is described in detail, this learning is a form of STDP learning with normalisation, $STDP + N$ learning. This learning mechanism is then applied to both individual spiking neurons and networks of recurrently connected spiking LIF neurons in a variety of different experiments designed to further the knowledge in the field, of what spiking neurons can learn, how well they can learn it and how this learning may possibly be improved. A metric of spike train similarity is introduced in chapter 6 and used extensively throughout the experimental work of this thesis. Chapter 7 demonstrates a novel application of $STDP + N$ learning, $STDP + N_{Type2}$ which, is shown to be capable of learning certain firing patterns that the standard application of $STDP + N$ learning, $STDP + N_{Type1}$ cannot learn.

Chapter 8 contains the conclusions to the original work presented in this thesis, along with suggestions for further, related future work that is based on the results of this thesis.

Part I

Background Research

Chapter 2

The Liquid State Machine

The Liquid State Machine (LSM) concept was first introduced in Maass *et al* (2002a), and also investigated extensively in Maass *et al* (2002b, 2004a, 2004b, 2005), Bertschinger and Natschläger (2004), Natschläger *et al* (2002a, 2002b), among others. It is a relatively new and quite novel attempt to model some of the observed structural, behavioural and computational aspects of the human neocortex. The Liquid State Machine essentially acts as a filter on a set of continuous input spike trains and enables this input to be represented in a higher dimensional form. The LSM in one of its basic forms could typically consist of a three dimensional column of highly recurrently interconnected spiking neurons, some of which receive continuous input spike trains from a separate pool, or pools, of spiking input neurons. The model used for the recurrent neurons is typically the *Leaky Integrate and Fire* (LIF) model (Gerstner and Kistler, 2002). Typically, the emphasis of LSM research tends to the investigation of the computational power of learning, utilising groups of neurons. This need to simulate groups of neurons rather than just individual neurons requires that the neuron models need to allow for the simulation of these recurrent networks in a reasonable timeframe and so the simpler LIF model, instead of the more complex ion channel models, is the most favourable.

The recurrent LSM network or 'microcolumn', (Maass *et al*, 2002a), typically

receives a temporally varying input $n_{input}(t)$ from a pool (or pools) of input neurons that are connected to a subset of the recurrent neurons within the LSM; an input pool comprises one or more spiking input neurons. The LSM acts as a medium through which the input can be expressed in a higher dimensional form — the number of neurons and the highly recurrent connections of the LSM, each with its own synaptic weight and time delay (among other parameters which are discussed later in section), make this possible. The LSM concept is essentially equivalent to the concurrently and independently developed concept of the Echo State Network, ESN, introduced in Jaeger (2001) and investigated further in Jaeger (2002). From an implementation point of view, the two approaches are the same and the reason for using the LSM as the basis for the work in this thesis is simply down to the abundance of literature dealing with the LSM, coupled to the availability of the necessary LSM simulation software CSIM (IGI Group, 2008) which runs under MATLAB.

In the context of the LSM, a readout neuron which receives a connection from all of the neurons within the LSM is then able to be taught, using a learning algorithm to modify the synaptic weights on the connections from the LSM, to perform some computable function F on the input ($n_{input}(t)$), with the strong limiting factor on the complexity of the filter F being the size of the LSM (Bertschinger and Natschläger, 2004). An example of this type of basic LSM setup can be seen in figure 2-1.

Using such a diverse representation of the input, it is possible to represent highly salient aspects of the input, which can then allow for the computation of complex functions on the input. Such functions include both linear and non-linear functions such as, for example, the training of a group of readout neurons on memory capacity functions — essentially n-bit parity tasks — or the XOR function, (Bertschinger and Natschläger, 2004), which will both be discussed in a later section.

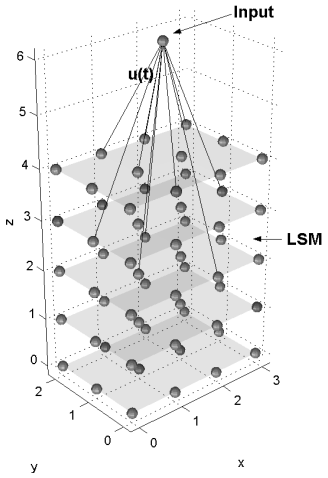


Figure 2-1: A Basic LSM consisting of LIF neurons with spiking input neuron n_{input}, generated using CSIM, IGI group (2008).

2.1 A Typical Implementation

Analyses of the computational qualities of the LSM typically involve the use of a group or column of highly recurrently connected LIF neurons which facilitate the projection of an input into higher dimensions, as shown in figure 2-1.

The connectivity between LSM neurons is generated such that those neurons that are spatially close together have a higher probability of forming a synaptic connection than neurons that are spatially more distant from each other. For example, if the term for the probability that two neurons are connected is given by, $e^{D(a,b)/\lambda)^2}$, then the greater the proximity of any two neurons then, the greater the probability that they will be connected. Here, λ determines the average distance between neurons as well as the average number of connections. The euclidean distance between two neurons a and b is given by $D(a, b)$.

Generally, the LSM is implemented in such a way that there exists only a single connection between any given pair of neurons, with a typical network neuron receiving hundreds of input connections from surrounding network neurons. Given that in a real structure such as the neocortex a single neuron could re-

ceive $10,000$ synaptic connections (Peters and Jones, 1984), this could be a short-coming of these typical LSM implementations, especially given that there has been little research into the effect on computational power that such highly multiple interconnected systems posses. It is unlikely that one can achieve a thorough understanding of something as complex as a neocortical column if one does not appreciate, and take into account, the scale of its complexity.

The universal computational power of the LSM concept has been successfully demonstrated in Maass and Markram (2004). It was shown by Maass and Markram that certain classes of LSM are capable of universal computation in an online scenario and on functions of time, or time series data. In other words, it was shown that within the LSM framework, and for any given continuous function of time, on a finite interval, it is possible to train a readout neuron of an LSM to approximate the function.

Additionally, it was shown that, in principle, for any Turing machine, an LSM can be created, whose readouts may be trained to produce all computations possible for that Turing machine.

The ability of the LSM to act as a high dimensional representation of its input is indeed a useful starting point for investigating the general computing power of micro-columns of neurons. However, research involving the LSM has typically involved using static synaptic weights within the recurrent network, as opposed to synaptic weights that be altered via a learning mechanism.

The detailed analyses performed in Maass *et al* (2002a, 2002b, 2004a, 2004b, 2005), Pfister *et al* (2006), Toyoizumi *et al* (2007), Bertschinger and Natschläger (2004), Natschläger *et al* (2002a, 2002b) and Jaeger (2001, 2002), on the LSM or the ESN, provide the neural network community with several interesting and important concepts and proofs concerning the behaviour and computational power of LSMs and LSM-type networks. The introduction of these concepts have established the LSM paradigm as a new and exciting tool for the community. What follows is a description of the most important aspects of the concepts arising from the core LSM research so far.

2.2 Learning within the LSM paradigm

Typically, the process of learning, in the context of the standard LSM paradigm, is accomplished as follows. An input stream $n_{input}(t)$ which, could consist of a single or many spike trains is presented to a subset of the recurrent LSM. The learning of a particular function on this input stream $F(n_{input}(t))$ is then accomplished by altering the synaptic weights that connect a readout pool of neurons — or even just a single neuron — to the recurrent network. Generally, the neurons which belong to the readout pools are fully connected to the LSM. This means that a single readout neuron receives a single connection from each of the recurrently connected neurons within the LSM.

The individual synaptic weights on the connections from the LSM to a readout neuron are modified by a form of gradient descent known as the p-delta rule — a variation of the delta rule (Auer *et al*, 2002), and is discussed below.

Using this technique, it can be said that a single readout neuron is basically treated as simple perceptron and that a group of readout neurons essentially functions as a committee. The output of each element of the committee is treated as a binary signal and the output of such a committee at a particular instant in time being regarded as the state (either firing or not) of the majority of the readout neurons which constitute the committee.

2.3 The p-delta rule

The parallel-delta (p-delta) rule (Auer *et al*, 2002), is a variation of the classic delta learning rule that was developed by Widrow and Hoff (1960). The essential difference between the two is that the p-delta rule modifies the delta rule in such a manner that it is applied to a committee of perceptrons with binary output that are arranged in parallel. Such a committee can receive input from a LSM and be trained using p-delta, to perform some function on the state of the LSM. The weight update rule for p-delta learning can be briefly summarised as follows.

The p-delta rule is not used in the original work of this thesis, it is shown here merely for completeness, as it is used heavily in the existing LSM research.

$$\alpha_i \leftarrow -\eta(||\alpha_i||^2-1)\alpha_i + \eta \begin{cases} (-\text{z}) \text{ if } \hat{\text{o}} > o+\epsilon \text{ and } \alpha_i \cdot z \geq 0 \\ (+\text{z}) \text{ if } \hat{\text{o}} < o-\epsilon \text{ and } \alpha_i \cdot z < 0 \\ \mu(+\text{z}) \text{ if } \hat{\text{o}} \leq o+\epsilon \text{ and } 0 \leq \alpha_i \cdot z < \gamma \\ \mu(-\text{z}) \text{ if } \hat{\text{o}} \geq o-\epsilon \text{ and } -\gamma < \alpha_i \cdot z < 0 \\ 0 \text{ otherwise} \end{cases}$$

Where α_i is the weight vector of a single readout neuron with i incoming connections, z represents the activity on each connection to the readout neuron, o is the actual output of the readout neuron, $\hat{\text{o}}$ is the desired (or goal) output of the readout neuron, η is the learning rate, γ is the margin around the origin which is kept clear of dot products (for stability), μ controls the influence of a particular weight update, ϵ is the acceptable error of the actual compared to the desired output of a readout neuron being trained. This is not, by any means, meant to be a full description of p-delta, but simply a brief summary. For full details see Auer *et al* (2002).

p-delta, although not used here, has been compared to several good machine learning algorithms such as: Gorman-Sejnowski (1988), Sigillito *et al* (1989), and Ster-Dobnikar (1996), on a variety of datasets and has been found to be of comparable performance, see Auer *et al* (2002). Also, Maass *et al* use p-delta to train pools of readout neurons connected to a LSM, see Maass *et al* (2002a). Additionally it should be noted that the performance of p-delta is highly dependent upon the choice of the parameters of the update rule. In actual fact, for best performance it is necessary to implement dynamic parameters — see appendices in Auer *et al* (2002).

2.4 Fading Memory

Fading memory is a concept introduced in Maass *et al* (2002a), and in Jaeger (2001) in which it is referred to as the *Echo State Property*. Informally, a LSM

is said to exhibit or posses the *fading memory* property if the *significant* components of its output spike train $y(t)$ at the current time t, are dependent only on its input spike train $u(t)$ over some finite interval into the past $[-T, 0]$. Therefore, *fading memory* can also be thought of as describing a LSM whose dynamics are *well behaved*. This means that the spiking dynamics are not driven by internally originating spiking sources but are primarily influenced by the driving input stimulus spike train and that crucially, the effect of the input spike train $u(T)$ at some point in the past T, on the output of the LSM $y(t)$ at some subsequent time t, where $t > T$, is increasingly diminishing as $(t - T) \rightarrow \infty$.

Consider a LSM at a time t with an input stream at this time given by $u(t)$ and an output stream $y(t)$, where $y(t) = Fu(t)$. F is a filter function that a readout neuron performs on the spiking activity of the LSM that is generated by the input spike train u(t). The formal definition of fading memory can be found in Maass *et al* (2002a).Fading Memory has two consequences for the LSM: i) It is not necessary in an LSM with fading memory to know anything about the input $u(t)$ beyond a finite interval $[-T, 0]$ into the past; ii) It is not necessary to know *precisely* what the input $u(t_1)$ was at any time t_1 in order to obtain the current significant components of the LSM output $y(t_2)$, where $t_2 > t_1$.

2.4.1 Fading Memory Experiments

Consider a LSM, M, that is connected to a single readout neuron which receives an input from each of the neurons in M. The readout is trained to learn some filter function of the spiking activity of the LSM. In this LSM setup, there can be considered to be two filter functions to consider. The first is the filter function performed by the LSM itself on the input spike trains it receives. This filter function is denoted by L^M (if we are to remain true to Maass' notation). The output of this filter function at time t, is the internal state of the LSM — a function which incorporates the firing states of all neurons within the LSM — and is given by $x^M = L^M u(t)$. The second filter function to consider is that performed by the readout neuron or, what is referred to in Maass *et al* (2002a)

as a memoryless readout unit. The readout is memoryless from the point of view that it does not remember or care about its input (the internal state $x^M(s)$ of the LSM) for any previous time s if $s < t$ where t is the current time. It is the filter function performed by the readout neuron on the internal state of the LSM that is the focus of much of the work undertaken on the LSM in Maass *et al* (2002a, 2002b, 2004a, 2004b, 2005), Bertschinger and Natschläger (2004), Natschläger *et al* (2002a, 2002b).

In order to illustrate the Fading Memory concept experimentally, the concept of the *memory capacity* of a LSM can be employed. A discussion of memory capacity follows in section 2.5.

2.5 Memory Capacity and Mutual Information

2.5.1 Information Theory and Mutual Information (MI)

For the sake of completeness and due to the use of memory capacity as a measure of the computational power of LSMs in Bertschinger and Natschläger (2004) and Maass *et al* (2002a) a brief introduction to information theory and mutual information is included. In 1948 Claude E. Shannon published his seminal paper on Mutual Information (Shannon, 1948). This paper is considered by most to be invaluable to the creation of the field of Information Theory and some its most fundamental concepts. Perhaps one of the most important concepts to be introduced in this work is that of the *Shannon entropy*. Entropy is a measure of the amount of disorder in a system and can also be thought of as a measure of the amount of *information* contained in a system.

If one considers two information sources to be both discrete and random, then the Shannon entropy can be used to describe how much information one source contains about the other. The Shannon entropy gives rise to the concept of the mutual information between the two sources. These sources have the potential to be thought of as almost any kind of information channel and therefore, the

measure of mutual information, MI, is a useful metric that has been applied in the fields of language (Magerman *et al*, 1990; Drábek *et al*, 2000; Zhang *et al*, 2006), telecommunications (Peacock and Collings, 2003; McKay *et al*, 2006), and computing/mathematics (Haussler and Opper, 1995) among many others.

In order to define and explain MI, first it is necessary to describe and explain the concept upon which it is based i.e. the Shannon entropy.

2.5.2 Shannon Entropy

The Shannon Entropy, denoted by H, is a quantity which represents the uncertainty of a variable that has a finite number of possible occurrences. For example, consider a set of events with probabilities $(p_1, p_2, ..., p_n)$. H is a quantity that reflects the certainty of the outcome. It can be seen that the scenario with maximum uncertainty is where the probability of each event is equal — in this scenario, the entropy, H, is at a maximum. If all events but one have a probability of occurring of 0 and one event has probability of 1 then we can be absolutely certain of the outcome and so uncertainty, the entropy, H, is zero.

Uncertainty is one way of looking at the quantity H, however H can also be thought of as representing the amount information we *know* about a set of possible events.

The three criteria that Shannon (1948) shows that H should satisfy are:

i) H should be continuous in the p_i.

ii) Given equal values for all p_i and n events, $p_i = 1/n$ and H should be a monotonically increasing function of n i.e. with more events there is more uncertainty, with events of equal probability.

iii) If a choice is split into two successive choices then the original H should be the weighted sum of the individual values of H. The original choice is denoted as: $H(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}) = H(\frac{1}{2}, \frac{1}{2}) + \frac{1}{2}H(\frac{2}{3}, \frac{1}{3})$

From Shannon (1948): $H(X) = -K \sum_{i=1}^{n} p_i \log p_i$ which, is the only form of H

that satisfies the above three criteria, where K is a positive constant. Given two possible events, one with probability p and the other with probability $q = 1-p$, a plot of H versus p can be obtained and is shown in figure 2-2. Using a logarithm of base 2 means that unit of H is the *bit*.

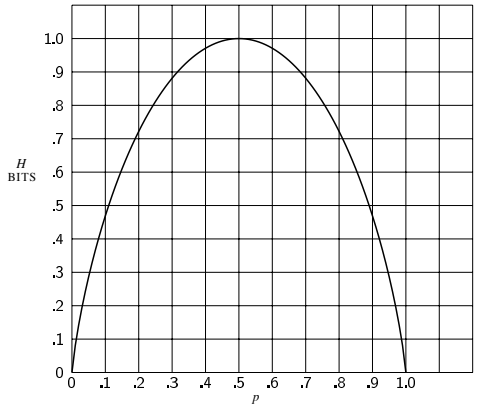


Figure 2-2: A plot of the Shannon entropy H in *bits*, *versus* the probability of each event, p. (Shannon 1948, p.20)

Shannon continues to show from figure 2-2 that H has several other properties:

1) $H = 0$ if and only if all p_i but one are zero and the non-zero p_i is 1 (condition i) above) — otherwise H is positive.

2) For n events $H \leq \log n$ and is maximum when all p_i are equal — the most uncertain situation (condition ii) above).

3) Consider two events x and y, with m and n possibilities respectively. If $p(i,j)$ is the joint probability of getting i for the first event and j for the second then the entropy of the joint event is given by $H(x,y) \leq H(x) + H(y)$ with equality if the events are completely independent.

4) Any change to equalise the probabilities results in an increase or no change at all in H.

5) Consider a similar situation to that in iii) but in which the two events x and y are not necessarily independent of each other. Shannon shows that the conditional probability $p(j|i)$ of getting j from the second event y given that the

first event x produced i is given by:

$$p(j|i) = \frac{p(i,j)}{\sum_j p(i,j)}$$

.

Shannon then defines a *conditional entropy* $H_x(y)$ of y which is the average of the entropy of y for each value of x, weighted by the probability of getting that particular x.

$$H_x(y) = -\sum_{i,j} p(i,j) \log p(j|i)$$

By substituting $p(j|i)$ with $\frac{p(i,j)}{\sum_j p(i,j)}$, Shannon obtains the relationship

$$H(x,y) = H(x) + H_x(y)$$

vi) The uncertainty of y is never increased with knowledge about x. Even if the two events are independent the uncertainty remains the same. Shannon's final condition for H is therefore given by: $H(y) \geq H_x(y)$.

To apply this to the quantity of mutual information between two events or information sources, consider two discrete and random information sources X and Y. Suppose that the Shannon entropy of X and of Y are given by $H(X)$ and $H(Y)$ respectively.

The mutual information between X and Y is the amount of information, for example, that Y contains about X and is given in Shannon (1948), as $I(X,Y)$, and it was shown that this can be expressed in terms of the Shannon entropy by the relationship:

$$I(X;Y) = H(X) - H_X(Y).$$

From the previous discussion of the meaning of the Shannon entropy, this expression can be considered to mean that the mutual information between X and

Y can be thought of as the uncertainty in X minus the uncertainty in X that remains when Y is known, i.e. the more information Y contains about X then, the smaller the uncertainty of X will be when Y is known. Therefore we subtract this smaller amount from $H(X)$ and consequently end up with a greater amount of mutual information between X and Y, $I(X;Y)$ than if Y was not known or if X and Y are independent of each other.

2.6 Memory Capacity

Memory capacity, a concept introduced in Bertschinger and Natschläger (2004), is a measure of how much information about its past inputs a system, such as the LSM, is able to encode in its current dynamical state at any time. Essentially, how much it can remember. Memory capacity is based upon the concept of *mutual information*, MI, as formulated in Shannon (1948). For most computational systems it is highly useful if not essential that the system has a memory — the larger and more robust the memory, the more information about its past can be stored. In the following description of the 3-bit parity task, consider the spikes contained in an input spike train $n_{input}(t)$ to be collected into time bins. The duration of these time bins should ideally be small enough such that it is likely that a single time bin will only contain one spike. It will be possible for a time bin to contain more than one spike however, a time bin will be treated the same whether it has one or more than one spike.

2.6.1 The 3-bit parity task and the LSM

Consider the basic LSM previously described and shown in figure 2-1. Suppose that this network is in some random, initial dynamic state of firing. The network is then given an input spike train that will drive the dynamic state of the system. Many systems that are considered to be computationally useful will need to have some memory of their input up-to some finite distance into the past. This finite duration puts a limit on the memory capacity of the system and it is this memory

capacity that the parity task is designed to measure.

The memory capacity of a LSM can be defined by means of the simple 3-bit parity task. Suppose we want to know the MI between the spike trains of neurons n_{input} and n_{output} — a measure of how much *information* the spike train of n_{output} contains about the spike train of n_{input}. Assume the input spike train to the network is given by $n_{in}(t)$, the output spike train is $n_{out}(t)$ and that the present time is denoted by T, so that $t <= T$.

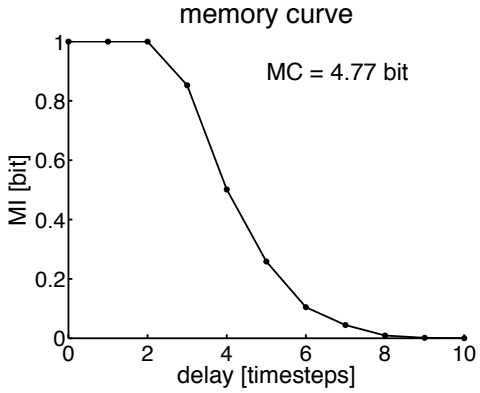


Figure 2-3: An example of a plot of $MI_\tau(n_{input}; n_{output})$ for various values of τ. The area lying beneath the line plot represents the memory capacity of the LSM. (Bertschinger and Natschläger, 2004, p.1427).

Define τ to be the time-step of the experiment and equal to the width of a time-bin of the input/output spike trains. Consider three successive time-bins of the input spike train at multiples of τ into the past. The memory capacity can then be defined by a simple 3-bit parity task as follows. The target output function which the readout neuron n_{output} is to learn, is determined by $F(n_{in}(t-\tau))$, where F represents the 3-bit parity task filter function on the input spike train and t is the current time. The mutual information between the actual output spike train of the readout neuron n_{output} and the desired output spike train, given by the training filter function $F(n_{in}(t-\tau))$ gives the value of $MI(\tau)$, where the input and output spike trains are considered over an interval $[(0+\tau), T]$.

For the 3-bit parity task where, the input to a LSM network is given by $n_{in}(t)$, the output of a readout trainable neuron is given by $n_{out}(t)$, and the desired output is

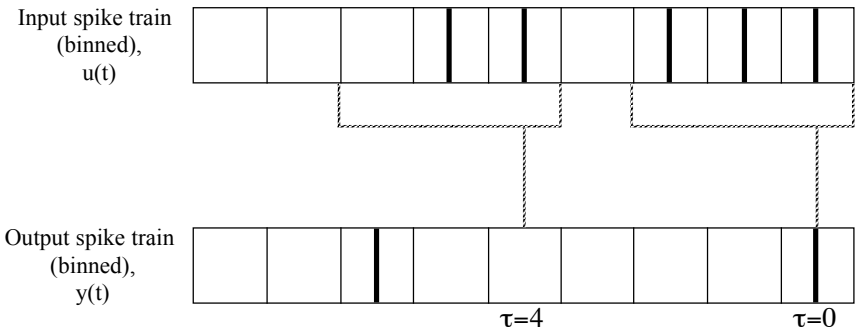


Figure 2-4: An illustrative figure to demonstrate the 3-bit parity task on a pair of input and output spike trains. In the time-bin at $\tau = 0$, the desired output parity function $y_\tau(t)$ contains a spike as each of the previous 3 time-bins of the input spike train contained at least 1 spike. Whereas the time-bin of the output spike train at $\tau = 4$ does not contain a spike, as parity does not exist between the previous three time-bins of the input spike train.

given by $y_\tau(t)$, we require that $y_\tau(t) = PARITY(n_{in}(t-\tau), n_{in}(t-\tau-1), n_{in}(t-\tau-2))$. If τ is taken to be zero (no delay), this means that the target spike train $y_\tau(t)$ will contain a spike at the current time t if the input spike train $n_{input}(t)$ has either at least one spike in each of the last 3 time bins or no spikes at all in each of the last 3 time bins. For clarity, this is illustrated in figure 2-4. The *MI* between $n_{out}(t)$ and $y_\tau(t)$ is then calculated for increasing values of delay τ — the value of τ is equal to the duration of a single time bin — and the memory capacity of a network can then be given by:

$$MC = \sum_{\tau} MI_\tau(n_{out}(t), y_\tau(t))$$

What this means is that the memory capacity of the LSM can be considered to be the sum of the *mutual information* between the output spike train of the network as given by $n_{out}(t)$ and the desired output spike train as given by $F(n_{in}(t-\tau))$ for increasing values of delay τ.

For increasing values of τ, the MC has been shown to drop off to zero, shown in figure 2-3, taken from Bertschinger and Natschläger (2004). This result can

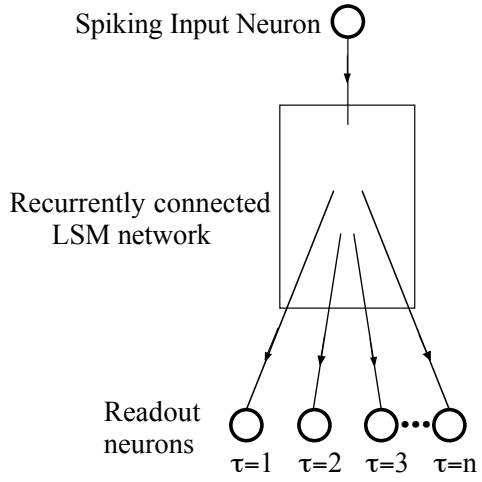


Figure 2-5: An illustration of a typical experimental setup that could be used to demonstrated the concept of fading memory.

be considered to be due to the fading memory concept discussed previously. The experimental setup required to perform the described MC tasks, and to obtain the memory capacity *curve* of a network is shown in figure 2-5. In this illustration each of the readout units would be trained to calculate the $PARITY$ function with a unique integer value of τ. In this way a plot of the mutual information between the actual output of each readout and the desired output versus the value of the delay τ can be obtained. This plot is shown in figure 2-3, and clearly demonstrates the drop off in MI between actual and desired output of the readout with increasing time delay τ. The drop off is entirely due to the memory capacity limit of the LSM which, is only able to hold a finite amount of information about its input history within its internal dynamics. The actual value of the memory capacity of a network can now be thought of as the total area encompassed by the MI vs τ plot and the τ axis.

2.7 Ordered, Critical and Chaotic Dynamics

The ability of the previously mentioned *readout neurons* to learn a particular function of the input stream of spike trains to the LSM is *heavily* dependent upon the response to the input stream, of the LSM itself. More precisely, performance of such simple, linear readouts is dependent on the LSM exhibiting neither too chaotic nor highly ordered dynamics in response to the input stream Bertschinger and Natschläger (2004).

The terms *ordered dynamics* and *chaotic dynamics* are defined in Bertschinger and Natschläger (2004), and this work stems from that of Derrida (1987). Broadly speaking, a network may be considered to exhibit ordered dynamics if the firing activity of the network is highly input driven, and the spiking activity produced by the neurons of the LSM itself do not become the main driving stimulus of the LSM over and above the input. Along a similar line of thinking, a network is considered to exhibit behaviour known as chaotic dynamics if the internal network activity generated by the network itself becomes the dominant, driving stimulus over and above the stimulus provided by the input stream. A chaotic network is highly influenced by the initial conditions of the network activity — initial internal states of recurrent neurons for example — rather than the input source. Additionally, a network is considered to exhibit critical dynamics if the dynamics are on the border between ordered and chaotic. The definition of this boundary for this criticality region is discussed below.

In Bertschinger and Natschläger (2004), the emphasis is investigating the computational performance of LSMs which exhibit ordered, chaotic and critical dynamics in terms of the parameters K — the number of incoming connections per neuron, σ^2 — variance of non-zero weights and $\bar{u}$ — the mean of the input stimulus. Together, K and σ^2 determine the connectivity structure of the recurrent network.

The Network state of a LSM is defined in Bertschinger and Natschläger (2004) as follows. All neurons within a LSM are treated as simple threshold gates whose internal state at a time t is given as $+1$ if it is firing and -1 if it is not firing.

Therefore, the state of a neuron i is defined as $x_i = \{-1, +1\}$. Also, i receives non-zero weights from K neurons within the LSM, with these weights being drawn from a gaussian distribution of zero mean and a variance given by σ^2. Therefore, the network state of an N neuron LSM is given by $\bar{x}(t) = (x_1(t), ..., x_N(t))$. The update rule for the state of a single neuron i is then given by:

$$x_i(t) = \Theta(\sum_{j=1}^{N} w_{ij} \cdot x_j(t-1) + u(t))$$

In which $\Theta(h) = 1$ if $h \geq 1$ else $\Theta(h) = -1$.

Consider a recurrently connected LSM with a subset of its neurons receiving an input spike train stimulus given by $u(t)$. Suppose that the network can have one of two different states of initial internal activity, denoted by $\bar{x}_1(t)$ and $\bar{x}_2(t)$. In each of these initial scenarios, suppose that the network is subjected to the input $u(t)$. In the first instance the initial network state is given by $\bar{x}_1(t)$ and in the second instance the initial network state is $\bar{x}_2(t)$; the new network states after exposure to $u(t)$ are given by $\bar{x}_1(t+1)$ and $\bar{x}_2(t+1)$ for initial states $\bar{x}_1(t)$ and $\bar{x}_2(t)$ respectively. The normalised Hamming distance between the two initial states $\bar{x}_1(t)$ and $\bar{x}_2(t)$ is calculated and compared to the normalised Hamming distance between the two network states, $\bar{x}_1(t+1)$ and $\bar{x}_2(t+1)$, after being presented with the input $u(t)$. If the distance between states has increased then the network is said to inhabit the chaotic phase — the dynamics are dependent on the initial state of the network. Whereas, if the distance has decreased the network is said to inhabit the ordered phase — dynamics are driven by the input stimulus, and differences caused by the initial network states $\bar{x}_1(t)$ and $\bar{x}_2(t)$ have died away over time.

The normalised Hamming distance between two network states $\bar{x}_1(t)$ and $\bar{x}_2(t)$ at time t is given by Bertschinger and Natschläger (2004) as:

$$d(t) = |\{i : x_{1,i}(t) \neq x_{2,i}(t)\}|/N$$

Where $x_{1,i}$ denotes the i^{th} component of the network state vector $\bar{x}_l(t),\ l = 1, 2$. The normalised Hamming distance between the two network states at time $(t+1)$

is denoted as $d(t+1;u)$ for a network that is presented with input u. This is related to the normalised Hamming distance between the original two network states at time t by the equation:

$$d(t+1;u) = \sum_{c=0}^{K}(K/c) \cdot d(t)^c \cdot (1-d(t))^{K-c} \cdot P_{BF}(c,u)$$

Where, P_{BF} is the probability that, of the K inputs to a single neuron, flipping the binary output of c of one these K inputs will cause the neuron to change its own binary output.

Relating to previous work in Maass *et al* (2002a), Bertschinger and Natschläger (2004) observe that LSM networks which populate the ordered phase can be said to posses the fading memory property previously mentioned.

It was found by Bertschinger and Natschläger that in addition to the critical line being the border between ordered and chaotic dynamics, networks located on the critical line *may* exhibit the best computational performance on some fundamental tasks.

The location of the critical line on the phase plot of $\bar{u}$ vs σ^2 seen in figure 2-6, is defined in Bertschinger and Natschläger (2004) as:

$$r \cdot P_{BF}(1, \bar{u}+1) + (1-r) \cdot P_{BF}(1, \bar{u}-1) = 1/K$$

Where, P_{BF} is the probability that 1-bit flip out of the K total inputs to a neuron will result in the neuron producing a different output in the two network states and r is the probability that the input u is given by $u = \bar{u}+1$ and $1-r$ is the probability that $u = \bar{u}-1$.

2.7.1 The effect of criticality of computational power

By using values for K, σ^2 and $\bar{u}$ that generate networks that lie in one of the three dynamical regions it is shown in Bertschinger and Natschläger (2004), that

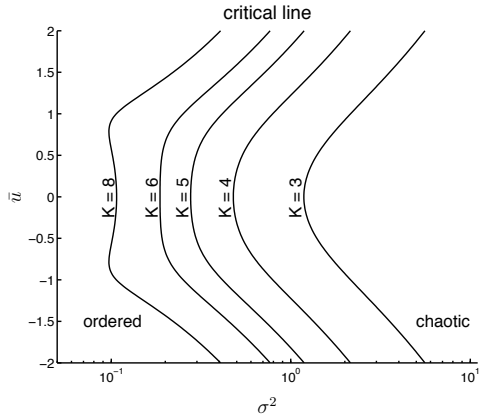


Figure 2-6: A plot of $\bar{u}$ against σ^2, showing the critical line, the border between ordered and chaotic network dynamics. (Bertschinger and Natschläger, 2004, p.1420)

choices of K, σ^2 and $\bar{u}$ that produce networks that exhibit critical dynamics are computationally better than both chaotic and ordered networks on two specific, fundamental tasks. Determining the computational power of a network is done using two types of task: i) Determining the memory capacity of the networks using 3 and 5-bit parity tasks; ii) Training a single readout neuron (linear classifier) that receives a connection from every neuron within the LSM to perform the XOR task upon the input to the LSM. Figure 2-7 from Bertschinger and Natschläger (2004), illustrates the claim of computational advantage of networks exhibiting critical dynamics, for the 3-bit parity task.

Having shown that the location of the critical line has been shown from plots of $\bar{u}$ vs σ^2 for different values of K (see above sub-section), Bertschinger and Natschläger demonstrate experimentally, the effect on the computational power of networks that inhabiting the critical phase has. This is accomplished primarily by using 3 and 5-bit memory capacity tasks. Consider the experimental setup outlined in the previous section on memory capacity and illustrated in figure 2-5. By performing the memory capacity test for many networks with $K = 2, 4$ and 8 and a selection of varying values for the parameters of σ^2 and $\bar{u}$, it is then possible to create a plot of the memory capacity of these network as a population density in the parameter space of σ^2 and $\bar{u}$. These results, obtained by Bertschinger and Natschläger, can be seen in figure 2-8.

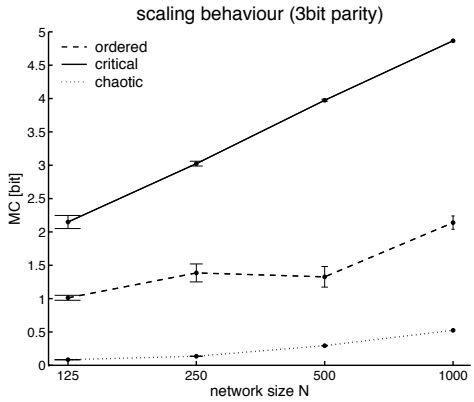


Figure 2-7: Plot of memory capacity, demonstrating the performance advantage on the 3-bit parity task of critical networks, compared to ordered and chaotic networks. (Bertschinger and Natschläger, 2004, p.1428)

It is shown that those networks which have the greatest MC value are clustered about the critical boundary line, with the best of these populating the ordered side of the critical line in the region where the variance σ^2 and the mean of the input $\bar{u}$ are relatively low valued.

As an aside, it should also be noted that in at least one subsequent paper (Maass *et al*, 2005) the usefulness of the criticality region as a performance determinant was questioned. Sufficiently robust predictors of performance which work well in all relevant scenarios are no doubt difficult to devise. However, is does appear that the criticality border does posses characteristics which mark it out as a logically reasonable gauge of the suitability of a network to be able to perform computationally well in a classical manner.

One can see however, that given the complexity of structures within the neocortex and the myriad structural types that could exist it may well be possible that beneficial structures may exist which enhance the computational power of other cortical structures without themselves being computationally powerful.

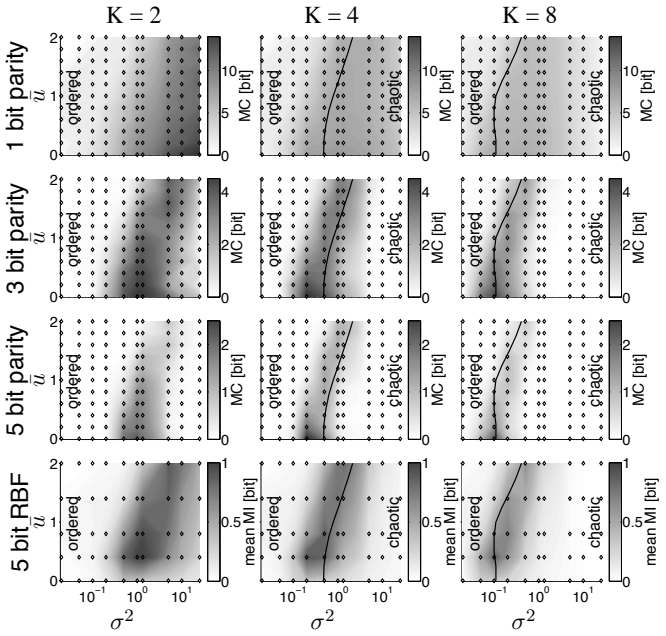


Figure 2-8: A plot of memory capacity, using 1, 3 and 5-bit parity tasks, as a population density in terms of σ^2 against $\bar{u}$, with $K = 2$, 4 and 8, as well as 5-bit RBF, each averaged over 10 networks. Darker areas indicate higher memory capacities, and therefore it can be seen that those networks with the highest memory capacities are clustered around the ordered side of the critical line. (Bertschinger and Natschläger, 2004, p.1429)

2.8 Structured LSMs

2.8.1 The Human Neocortex

The neocortex of the human brain is a structure that has evoked a huge amount of interest among researchers based in the field of artificial neural networks and neuroscientists who investigate the brain. The LSM is itself intended as a basic model for the behaviour of activity in the neocortex and in particular, behaviour within neocortical micro-columns (Jones, 2000; Mountcastle, 1997; Eccles, 1984). As such, it is impossible to convey adequately the wealth of work that has been produced on the subject in this introduction. What follows should be considered to be merely a basic and generalised overview of the observed structure, proposed

uses and computational capability of the neocortex.

Structurally, the neocortex comprises of, broadly speaking, around 6 layers (Braak and Braak, 1992; Peters and Jones, 1984) that may be considered distinct from each other in terms of: The synaptic connectivity structure, including synaptic density (Braak and Braak, 1992; Peters and Jones, 1984); The neuron types present within each of the layers (Braak and Braak, 1992; Peters and Jones, 1984); The afferent and efferent regions of the brain for each layer (Peters and Jones, 1984; Sereno *et al*, 1995; Kaas and Krubitzer, 1991)

The number of layers which make up a neocortical micro-column actually varies throughout the brain and is dependent on the region that is under consideration. There may actually be up to 9 distinct layers in certain brain regions. (Braak and Braak, 1992; Peters and Jones, 1984; Ramón y Cajal, 1902, 1955; Rose, 1926).

Typically the layers are labeled $I - VI$ with layer I being the topmost layer closest to the surface and layer VI being the deepest layer of the neocortex. Layer VI is sometimes referred to as being made of two distinct layers VIa and VIb. A basic rendering of a neocortical micro-column with its layers labeled, can be seen in figure 2-9.

The different layers of the neocortex consist of a variety of different neurons that vary in their function (Ferster *et al*, 1996), size, shape and, importantly, the characteristics of the connection which they form to the surrounding neocortical neurons (Peters and Jones, 1984; Sereno *et al*, 1995; Kaas and Krubitzer, 1991).

First, a brief word on the general structure of biological neurons. The core of the neuron itself is known as the *soma*. Branching out from the soma are structures known as dendrites which themselves have numerous branches and are known as *dendritic trees*. It is the dendrites that relay firing activity from connected neurons into the soma. The soma communicates its output via the *axon*. In general, a neuron will have only a single axon but this will eventually branch out into many dendrites which connect the neuron to many other neurons. The point at which an axon of one neuron interfaces with the dendritic tree of another neuron is called the synapse. The synapse is able to relay the electrical firing

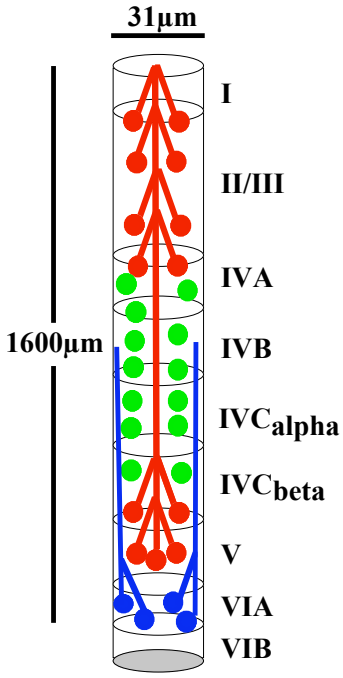


Figure 2-9: An illustration of the laminar detail of a neocortical column. Redrawn from DeFelipe and Fariñas (1992).

activity from the axon of one neuron to a dendritic tree of another neuron by means of complex chemical ion channels. Detailed analysis and descriptions of the varieties of neurons and their constituent structures can be found in (Cajal, 1902, 1955; Peters and Jones, 1984; Nimchinsky *et al*, 1995, 1999; Allman *et al*, 2002; Hof and Van der Gucht, 2006; DeFelipe and Fariñas, 1992).

Pyramidal neurons are relatively large bodied neurons whose name derives from the triangular appearance of their structure. Physiologically, a pyramidal neuron consists of a *soma* which contains the nucleus of the neuron and this receives input from a large dendritic tree and has only single (eventually branching) axon which relays the output of the neuron, (DeFelipe and Fariñas, 1992, p.570). It is the overall shape of the dendritic tree soma and axon, shown in figure 2-10 to which the triangular appearance previously mentioned, refers. Pyramidal neurons account for the majority of neurons within the neocortex and can form connections that range in length from short range connections to nearby neurons,

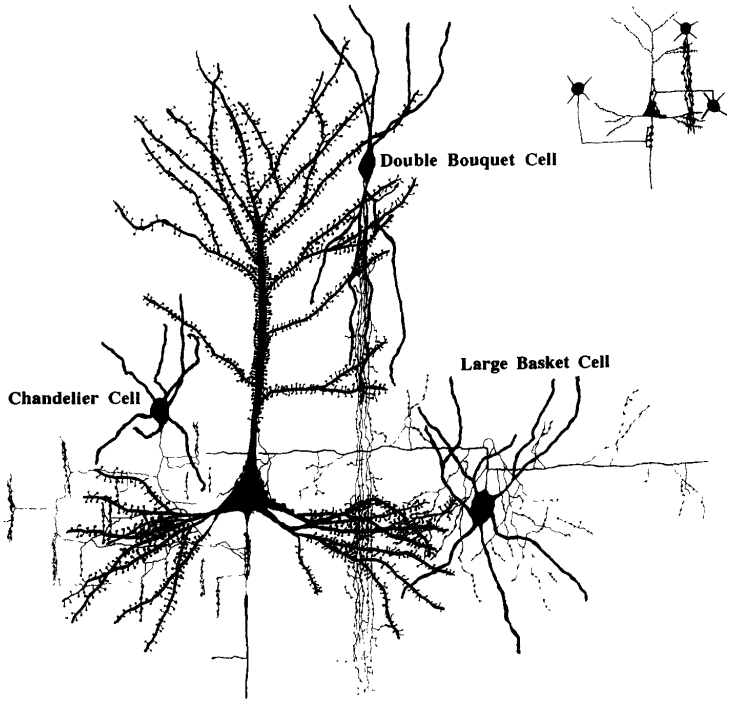


Figure 2-10: An illustration of a typical pyramidal neuron and some connectivity. (DeFelipe and Fariñas, 1992, p.570).

to relatively long range connections to neurons that, can span up to all of the layers of a micro-column and beyond (Peters and Jones, 1984). Connections from pyramidal neurons have been observed that are capable of spanning the two hemispheres of the brain (Peters and Jones, 1984). If one considers that the diameter of a micro-column is $30\mu m$, (Jones, 2000) it can be seen that connections from pyramidal neurons could also act as conduits that are able to relay information between relatively distant neural structures. Pyramidal cells can possess two different types of synapses known as Gray type I and Gray type II. These synapses are morphologically different and indicate and excitatory and inhibitory synapses respectively (Peters and Jones, 1984).

Among the other neuron types that are found within the neocortex are those neurons known as *spindle* neurons, (Nimchinsky *et al*, 1995, 1999; Allman *et al*, 2002; Hof and Van der Gucht, 2006). Spindle neurons are referred to as

such because, they only have a single input to the soma as opposed to the large dendritic tree inputs that are seen in the case of the pyramidal neuron. An illustration of a spindle neuron can be seen in figure 2-11. It is thought that their unique structure may make the spindle neuron a facilitator of higher cognitive abilities. They are among the most rare of neuron types and are only located within the region of the brain known as the *Anterior Cingulate Cortex* ACC and the *frontoinsular cortex*, additionally they are most abundant in humans and have only, so far, been found in the great apes, whales and dolphins, (Hof and Van der Gucht, 2006; Nimchinsky *et al*, 1999). In Nimchinsky *et al* (1999) it was found that the amount of spindle neurons in samples taken from layer V was 5.6% of the number of pyramidal neurons in the sample for humans and 2.3% for gorillas.

Figure 2-11: An image of a spindle neuron, showing the *cable-like* structure of the spindle neuron. (Nimchinsky *et al*, 1995, p.33)

The form of the structure of spindle neurons means that they appear like cortical cabling, linking brain regions directly, over relatively large distances (Nimchinsky *et al*, 1995). It is interesting to note that spindle neurons only appear to form in humans post-natally, and are attributed to intelligent behaviour. It is thought that spindle neurons within the ACC may serve to *refract* neural waves from one brain region to another, and that the ensuing neural interference patterns can be manipulated by the ACC to allow for error feedback and the ability to make *good*

choices for a given task, (Allman *et al*, 2002).

2.9 Laminar Structured Liquid State Machines

The typical LSM implementation will comprise a column of neurons that is generated with the same connectivity parameters throughout the entire column. In order to address this disparity between the basic LSM model and the observed laminar structure of actual biological micro-columns, relatively recent work has been carried out in Hæusler and Maass (2007), which examines the computational benefits of Liquid Sate Machines that posses an extra degree of sophistication over the standard LSM model. These modified LSMs are based upon the concept of a laminar structure discussed above (Hæusler and Maass, 2007).

The laminar LSM model employed by Hæusler and Maass consists of essentially 4 layers, each of which is generated using connectivity and structural data obtained from studies of the neocortices of both cats and rats (Anderson *et al*, 2000; Binzegger *et al*, 2004; Ferster, 1987; Hilgetag *et al*, 2000; Thomson *et al*, 2002). The focus of investigation of this work is to investigate the effect of a biologically inspired laminar columnar structure on the computational performance (XOR task) of a trained linear readout neuron, compared to simply using a randomly generated or *amorphous* recurrent network. It was shown experimentally by Hæusler and Maass that, readout neurons connected to networks that were generated using the biologically inspired laminar structure, were typically able to outperform those readout neurons that were connected to an amorphously connected network, by a statistically significant amount. This result is an indication that merely the introduction of connectivity variations across regions of a recurrent spiking neural network may provide more diverse firing patterns which, may in turn enhance the computational power of trained readout neurons that receive input from the network (Hæusler and Maass, 2007).

2.10 Limitations and Conclusions

As a model for investigating the computational properties of models of artificial neocortical columns the LSM is a highly satisfactory approach and one which has produced much insight and valuable results. However, if one wishes to investigate the more elusive properties of groups of recurrent neocortical neurons, such as memory storage and transfer, synaptic modification within neocortical structures and the relationships between this modification and the creation of memory then, one must discard the LSM approach (or at the very least modify it), in favour of a model which incorporates an allowance for long term modification of the efficacy of the synapses within the recurrent micro-circuit. Aspects of these elusive properties will be addressed in part II of this dissertation.

The LSM approach of using unchanging synaptic weights in the liquid layer does not allow for any investigation of how, for example, an input stimulus will affect these weights over a sufficiently long duration of exposure. Additionally, if it is one's goal to model such recurrent micro-circuits in as realistic a manner as possible then again, the LSM is insufficient. Static weights and linear regression learning regimes, while being computationally useful, are not a realistic representation of the learning that is known to occur within recurrent neural networks of biological neurons.

It is known from Bliss and Lømo (1973) and Cooke and Bliss (2006) that Long Term Potentiation (LTP) and similarly, Long Term Depression (LTD) are important processes in the formation of a network structure moulded by the synaptic firing activity that it receives. Without equivalent modification processes, any biologically realistic study of memory formation cannot be considered to be realistic.

Additionally, the treatment of readout neurons as simple linear perceptrons is also a limitation of the current LSM paradigm. The readouts are connected to *all* neurons in the LSM so, while this means that the readout will be able to sample the output of the entire LSM and therefore be better able to perform a given function upon the network state at any given time. This is an unrealistic scenario

which breaks with the idea that neurons are more likely to receive connections from their nearest neighbours and that the probability of connectivity decreases with neuron separation. At the same time the readouts typically do not have any kind of feedback connectivity to the LSM layer, and therefore the effect of such feedback on learning cannot be addressed in this scenario.

Synaptic weights are also allowed to 'flip' their sign — excitatory connections are allowed to become negative and *vice-versa* for inhibitory connections using the p-delta learning rule, although recently in Legenstein *et al* (2005) there has been some acknowledgment that this weight 'flipping' is unrealistic and the method of STDP learning outlined in this thesis uses a bounding and normalisation technique to ensure that weight flipping cannot occur.

The existing LSM research has produced a great deal of useful and important results, among them are the work performed to investigate the how the memory capacity of LSM networks varies with connectivity parameters, and the results demonstrating the computational power of the LSM micro-circuit.

Recent work (Legenstein *et al*, 2005; Pfister *et al*, 2006; Toyoizumi *et al*, 2007) that has been carried out concurrently with the work in this thesis, focusses on the effects of LTP and LTD on learning for single spiking neurons. However, there is much still unknown about the nature of memory formation and long term learning for both recurrent networks that consist of many neurons and individual neurons that receive many inputs. While the LSM provides a good basis for a starting point, it should be treated as such and expanded upon appropriately.

The work in this thesis, principally the work detailed in chapters 5, 6 and 7, aims to help further the understanding of some of the properties, behaviour and possible functions of networks of highly recurrently connected neurons in which a modified form of STDP learning is applied to the synapses of the recurrent network.

Chapter 3

Obtaining Precise Spike Trains

3.1 Introduction

Estimates for the number of neurons in the human neocortex, based on sampling of brain tissue, typically lie in the range 19 to 23 billion neurons (Pakkenberg *et al*, 1997, 2003). One could be forgiven for thinking that focussing attention on the computational capabilities of a single neuron in such a vast population would, perhaps, be a meaningless task. Indeed, in some instances it may make more sense to concentrate on investigating the computational power and behaviours of populations of neurons rather than individual neurons. Functional Magnetic Resonance Imaging (fMRI) scans of the active brain show entire populations/brain regions responding to certain specific stimuli. A detailed overview of such fMRI techniques can be found in Jezzard, Matthews and Smith (2003).

However, techniques for probing the electrical responses of the brain *in vivo*, are not yet sufficiently sophisticated to be able to observe neural activity on the scale of individual neurons and so, the response of single neurons that belong to such huge populations cannot yet be observed accurately *in-vivo*. Current experimental methods for collecting firing data from neurons make use of arrays of electrodes, known as Multiple-Electrode Arrays (MEA's). Experiments that use

the MEA technique for acquiring firing times from populations of real neurons, generate vast amounts of data. However, while the resolution of MEA technology is improving, it is still not possible to directly observe the activity of specific individual neurons within a larger population of neurons. Technical data pertaining to the capabilities of MEA technology that is currently available may be found in Alpha Med Sciences (2008). A typical MEA may have 64 electrodes, arranged in a 8x8 configuration, with the separation between electrodes being approximately $100\mu m$. A tissue sample containing a population of neurons, could then be placed on the MEA. One of the electrodes could be used to inject and input current into the tissue and the remaining electrodes can be used to record the firing activity response that is produced by the neurons throughout the tissue sample.

The MEA is limited to recording the current response at only $100\mu m$ intervals throughout a sample. Consider that a neocortical microcolumn has a diameter of $30\mu m$ and a depth of $1600\mu m$, and that this column may itself contain hundreds of cells, see figure 2 in Jones (2000, p.5020), this means that the spike trains recorded by a typical MEA cannot be considered to have been generated by an individual neuron but from the population of neurons that surround that particular electrode. However, it is possible that by using the recordings of many electrodes and by applying spike sorting algorithms that make use of principle component analysis, that individual spikes recorded by the MEA can be assigned to the correct source neuron. Studies of spike sorting can be found in Roberts and Hartline (1975), Abeles and Goldstein (1977) and in Chandra and Optican (1997) among many others. Spike sorting allows for not simply the separation of spike train currents from variable background currents within a sample of neural tissue, but also separation of individual spike trains and subsequently assignment to the correct source neurons from which a particular spike train originated. Spike sorting is made more complex when it is noted that the waveform of spikes from a particular single neuron is subject to change if, for example, two spikes are emitted from the same neuron in close temporal proximity. In such a scenario the waveform of the second spike would be affected by the first spike and therefore, potentially hindering the classification of the second spike as belonging to that neuron.

The work in this dissertation is concerned with simulated, artificial neurons and not real, biological ones. However, irrespective of whether real or artificial neurons are used, the study of precise spike trains and in particular in getting neurons to produce them accurately is relatively new. Far more research has been undertaken which investigates the computational aspects of networks of neurons — as was previously discussed in chapter 2. This does not mean that single neurons are not computationally capable elements in their own right, nor does it necessarily mean that individual neurons cannot or do not perform macroscopically important computational tasks in their own right.

An alternative way of dealing with spiking outputs from neurons is to consider their firing rates. However, studying the precise temporal nature of individual spike trains is necessary if one wishes to investigate the temporal *structure* of the spike trains between neurons and the information that this may convey. The temporal structure of a spike train merely refers to a spike train in which the precise placement of the constituent spikes relative to each other is important. This cannot be observed if one uses firing rates. If one were to consider only the firing rate and not the precise nature of individual spike trains, the detail contained in the spike train data becomes *blurred*, and lost to any subsequent analysis. Spiking neurons offer a method of encoding information sent between neurons in a completely temporal manner. As a result they can be used to study highly temporally precise data that cannot be done using simple perceptron-style learning.

Consider an artificial, simulated neural network that uses biologically realistic learning rules at the synaptic level. An interesting approach would be to demonstrate that a single neuron can respond to a *specific* input from a large population with a highly precise spike train. This could be a useful result in demonstrating that the precise output of single neurons may be of great computational use on a macroscopically functional level.

Additionally, it would appear to be logical that an important computational function left to be performed solely by a single lone neuron would be at great risk of being corrupted or lost completely if that neuron were to be damaged

or destroyed. For this reason, one may expect to see some form of redundancy inherent in neural structures, for example, a distributed population of neurons coding for the same function. Neocortical micro-columns may be a manifestation of just such a redundancy principle in action.

3.2 Optimised STDP Learning of Precise Patterns

The work by Pfister *et al* (2006) and Toyoizumi *et al* (2007), are examples of the current research being undertaken to investigate the effect of the application of a synaptic modification algorithm based upon gradient ascent derived learning rules. The research investigates the ability of a single spiking neuron to generate spike trains consisting of a specific sequence of precisely timed spikes, in response to a specific input spiking pattern. In other words, can STDP be used to make a neuron learn a precise temporal pattern? The work presented in Pfister *et al* (2006), investigates the form of the STDP learning function that produces the optimal solution for adjusting the input synapses to a single neuron in order for that neurons to learn a precise spike train.

The gradient ascent algorithm optimises the synaptic weights of the spiking neuron such that it will fire at specific desired firing times. It is claimed in Pfister *et al* (2006), that their training method does not show any unique reason for the presence of synaptic depression in response to a pre-synaptic spike arriving *after* a post-synaptic spike. It is shown that the existence and also the strength of synaptic depression can be attributed to specific parameters of the optimisation rule.

The results obtained by the group, using their algorithm are compared to the biologically inspired STDP learning scenario seen in Bi and Poo (1998, 1999, 2001), Markram *et al* (1997) and Zhang *et al* (1998), as well as the reinforcement learning method, Xie and Seung (2004), and Seung (2003).

3.2.1 Experimental Setup

The approach utilised in this work is to use, as a basis, a temporal coding scheme of the type seen in the human touch stimuli (see the work by Johansson and Birznieks (2004)). The ideas presented by Pfister *et al* (2006) could similarly be applied to other temporal coding schemes as seen in Carr and Konishi (1990), Bell *et al* (1997), Hopfield (1995), Brody and Hopfield (2003), Mehta *et al* (2002) and Panzeri *et al* (2001).

Consider an experimental setup as shown in figure 3-1 as described in Pfister *et al* (2006). The single post-synaptic neuron seen here could be replaced with multiple post-synaptic neurons, but in order to simplify the explanation only a single neuron is considered. The neuron receives input from N input channels or *input lines* that are denoted by j where, $1 \leq j \leq N$.

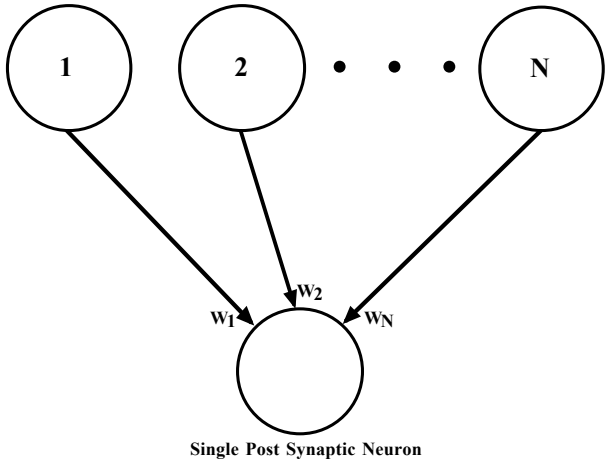


Figure 3-1: An illustration of a single spiking neuron which receives a single input from each of N input neurons/channels.

The model used for the post-synaptic neuron is a relatively standard one in which, a pre-synaptic spike at a firing time t_j^f will produce the Excitatory Post Synaptic Pulse (EPSP) within the membrane of the post-synaptic neuron.

The time course of this EPSP is described by the double exponential equation:

$$\epsilon(s) = \epsilon_0 \left[exp(-\tfrac{s}{\tau_m}) - exp(-\tfrac{s}{\tau_s}) \right] \Theta(s)$$

This equation is taken by Pfister *et al*, from Gerstner and Kistler (2002). Where, τ_m is the time constant of the membrane and is set to $10ms$ and τ_s is the time constant of the synapse and is equal to $0.7ms$. As a consequence the resulting rise time of the EPSP is $2ms$. $\Theta(s)$ is the Heaviside step function and is equal to 1 if $s \geq 0$ and 0 otherwise. The parameter ϵ_0 is set to a value of $1.3mV$. The reason that Pfister *et al* give for this is so that a spike that is elicited at a synapse whose weight is set to 1 will produce a spike with an amplitude of $1mV$.

In addition to receiving input from N synapses, the post-synaptic neuron also receives a 'teaching' input from a group of neurons. The function of this input is to stimulate the post-synaptic neuron to fire, or at least be more likely to fire, at a specific desired time denoted at t_{des}, which represents the precise input that the neuron is required to learn to perform.

The form of this teaching current, given in Pfister *et al* (2006), is a square current pulse described by:

$$I(t) = I_0 \Theta(t - t_{des} + 0.5\Delta T)\Theta(t_{des} + 0.5\Delta T - t)$$

Where, the amplitude is given by I_0 and duration of pulse is denoted by ΔT. The membrane activity that is produced by this teaching input is then given in Pfister *et al* (2006) as:

$$u_{teach}(t) = \int_0^\infty k(s) I(t - s) ds$$

Where, $k(s) = k_0 exp(-s/\tau_m)$ and in which k_0 is inversely related to the membrane capacitance and is a constant.

3.3 Gradient Ascent Learning Algorithm

In this model, learning takes place in the form of the modification of synaptic efficacies. The modification rule here is a version of gradient ascent and is given in Pfister *et al* (2006) as:

$$\Delta w_{ij} = \alpha \frac{\delta L}{\delta w_{ij}}$$

Where Δw_{ij} is the change in the synaptic efficacy (or weight), of the synapse connecting the pre-synaptic neuron j to the post-synaptic neuron i, α is the learning rate, L, is a quantity that represents optimality and is discussed below.

Instead of employing a deterministic approach, in which a thresholding method is used to determine when a given post-synaptic neuron fires, a probabilistic method is used. A full explanation of the stochastic model can be found in Pfister *et al* (2006), and is not discussed in full detail here.

In a deterministic model a post-synaptic neuron will generate an action potential if its internal state or membrane potential rises above a pre-determined threshold. Whereas in this and other stochastic models, a post-synaptic neuron will generate an action potential according to a point process in which the time dependent stochastic intensity is given by Pfister *et al* (2006) as:

$$\rho_i(t|x, y_t^i) = g(u_i(t|x, y_t^i))$$

Where $g(u) = \rho_0 exp(\frac{u-\nu}{\Delta u})$ in which $\nu = -50mV$, $\Delta u = 3mV$ and $\rho_0 = 1/ms$ and represent the threshold, the width of the threshold region and the intensity of stochasticity at the threshold value respectively. x is the input stimuli to the post-synaptic neuron and y_t^i is the recent firing history of the post-synaptic neuron. The stochastic intensity is dependent on these two variables because each have a refractory effect on the behaviour of the membrane potential. For example, if the post-synaptic neuron has only just fired a spike, then it will experience a refractory period during which it is less likely than usual to fire,

(Gerstner and Kistler, 2002). Similarly, if the neuron has just received some input spikes that were not quite of the magnitude necessary to elicit an action potential from the post-synaptic neuron then, for a short period of time the neuron will be more likely to emit an action potential to a subsequent input spike — due to the raised membrane potential experienced from the previous input activity.

It is noted in Pfister *et al* (2006) that, other forms for g may be used without affecting the results in an adverse manner, at least from a qualitative point and that for $\Delta u \rightarrow 0$ the stochastic model becomes equivalent to the deterministic LIF model, see Gerstner and Kistler (2002).

For the sake of brevity it is simply stated in this thesis that the optimality term L can be expressed as log-likelihood of the stochastic intensity term ρ. The full derivation and explanation can be seen in Appendix A of Pfister *et al* (2006).

3.4 Optimal Spike Timing Results

The results of this work by Pfister *et al* can be categorised into three separate and increasingly more complex scenarios A, B and C, each of which can be broken down further into an unconstrained and constrained scenario.

3.4.1 Scenario A

Scenario A is the simplest of the three scenarios and involves just two neurons. One of the neurons is a post-synaptic neuron Y and receives a single synaptic connection from a pre-synaptic neuron X.

In the unconstrained scenario, the post-synaptic neuron must fire a single spike at time t_{des} in response to a single spike at time, $t < t_{des}$, from the pre-synaptic neuron, whereas in the constrained scenario, the post-synaptic neuron must fire a spike *only* at time t_{des} and at *no other* time, in response to a single spike from the pre-synaptic neuron.

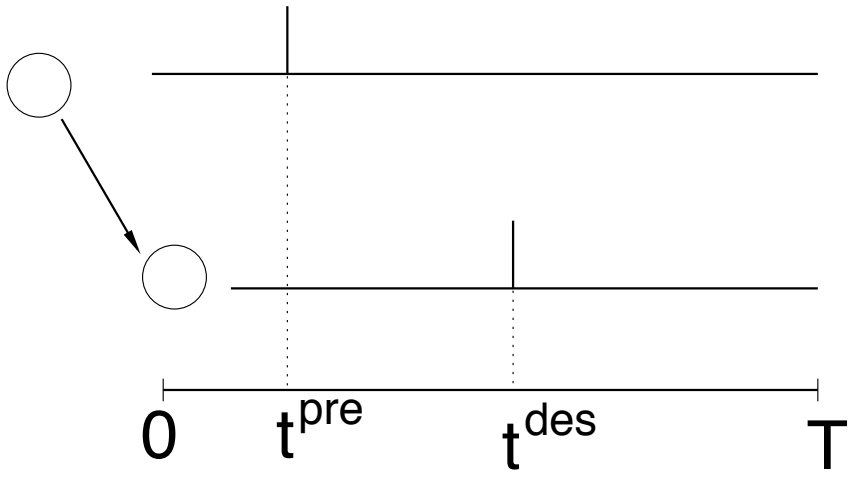


Figure 3-2: The pre and post-synaptic neurons of the Scenario A setup, taken from Pfister *et al* (2006, p.1326).

3.4.2 Unconstrained Scenario A (A_u)

The assumption is made that the post-synaptic neuron has not fired recently and so any refractory effects can be ignored. Additionally, instead of considering a sequence of input spikes, there is only a single input spike in this scenario. These two conditions mean that the expression for the stochastic intensity now becomes $\rho(t|x, y_t) = \rho(t|t_{pre})$.

The optimality term, L^{A_u}, for this scenario is then given as $L^{A_u} = log(\rho(t_{des}|t_{pre}))$ (Pfister *et al*, 2006), which allows Pfister *et al* to re-write the gradient ascent on the synapse mediating the connection between Y and X as:

$$\Delta w^{A_u} = \alpha \frac{\rho'(t_{des}|t_{pre})}{\rho(t_{des}|t_{pre})} \epsilon(t_{des} - t_{pre})$$

A plot of Δw^{A_u} against $\Delta t = t_{pre} - t_{des}$ from Pfister *et al* (2006), can be seen in figure 3-3. The plot is the mirror image of an EPSP and is, qualitatively at least, independent of the form of the function $g(u)$.

This is only a simple model for determining the optimal update rule and as such it has a fundamental shortcoming in that, if the update according to the gradient

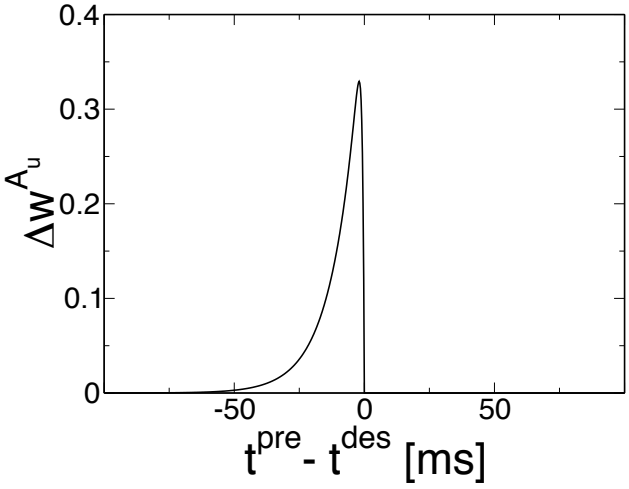


Figure 3-3: A plot of the change in synaptic weight w.r.t the difference in firing times between the pre and post-synaptic neurons. This figure is taken from Pfister *et al* (2006, p.1326)

ascent is calculated and applied iteratively, one would expect the perfectly optimal solution to converge, such that weight changes become zero i.e. $\Delta w^{A_u} = 0$. However, this is not what is observed in this particular model. Instead, Pfister *et al* observe that if $\Delta t < 0$ the solution tends to ∞ with continued iteration of the update rule and if $\Delta t \geq 0$, no unique synaptic weight is found. It is stated in Pfister *et al* (2006), that this is directly attributable to the fact that this particular model has no consideration of inhibitory synapses, only excitatory connections are modeled.

3.4.3 Constrained Scenario A (A_c)

The requirement for this scenario is intended by Pfister *et al* to simply allow for the incorporation of inhibition into the Scenario A model. Consider the experimental setup previously used in the unconstrained scenario A. The additional requirement now, is that the post-synaptic neuron fires at no time other than t_{des}. The optimality function for this scenario L^{A_c} is once again given by the log-likelihood of the post-synaptic neuron producing a spike at the desired time

t_{des} when given a pre-synaptic spike at a time t_{pre}.

$$L^{A_c} = \log(\rho(t_{des}|t_{pre})) - \int_0^\infty \rho(s|t_{pre}, t_{des})ds.$$

As a result, the synaptic update rule is given in Pfister *et al* (2006) as:

$$\Delta w^{A_c} = \alpha \frac{\rho'(t_{des}|t_{pre})}{\rho(t_{des}|t_{pre})}\epsilon(t_{des} - t_{pre}) - \alpha \int_o^\infty \rho'(s|t_{pre}, t_{des})\epsilon(s - t_{pre})ds$$

As before, this update function can be plotted against Δt, to give an STDP learning function as shown in figure 3-4 which is taken from Pfister *et al* (2006).

Pfister uses the learning function plot to demonstrate two separate cases. For the first case assume that, the post-synaptic neuron is not trained specifically to fire at time t_{des} i.e. the neuron may spontaneously emit a spike at t_{des}. If this happens, then the update rule is applied and the synaptic efficacy is modified accordingly. With each iteration of this method, the post-synaptic neuron will be more likely to fire spontaneously at time t_{des}.

For the second case consider that the post-synaptic neuron is forced, by some external teaching input of duration $1ms$, to fire at time t_{des}. It can be seen that the resulting STDP plot for different values of the reset potential η_0 demonstrate that, if the after-spike reset is zero, the STDP function has a strong depressive effect on synapses that are incident on the post-synaptic neuron immediately after post-synaptic spiking. The explanation given for this is that in the case where a pre-synaptic spike arrives after the post-synaptic spike, the resulting weight change is primarily controlled by the after potential of the membrane of the post-synaptic neuron $u_{AP}(t) = \eta(t) + u_{teach}(t)$. So when the reset term $\eta(t)$ is zero, the teaching input will totally dominate the weight change and a large depressive weight change is obtained. If, however the reset is small but non-zero, then this will go someway to reducing the effect of the teaching input on the after potential and the depressive effect will be less dominating.

The two cases considered in constrained scenario A show the differences to the synaptic update rule, between using the generative approach and that of using

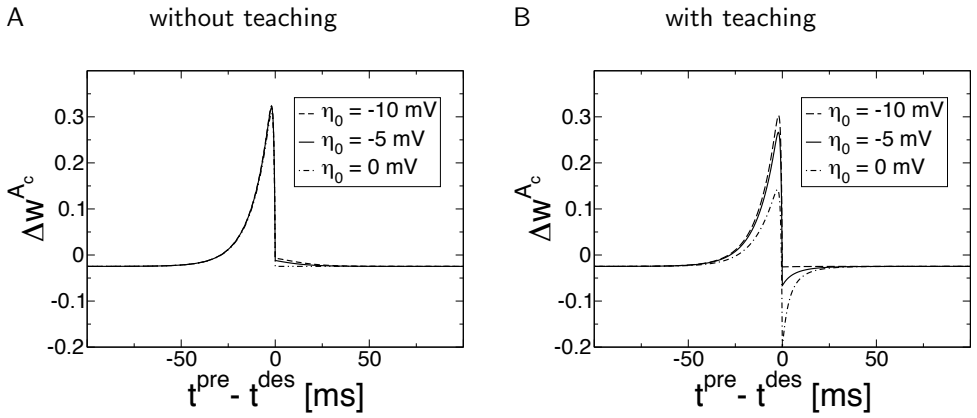


Figure 3-4: Plots of the resultant weight change vs. the difference between pre and post-synaptic firing times using the generative approach (A), and a teaching stimulus input (B). The figure is taken from Pfister *et al* (2006, p.1328).

a teaching input method. It should also be noted from the STDP function in figure 3-4 that there exists a negative offset of the functions along the y-axis. The presence of this offset is due to the condition that the post-synaptic neuron should not fire at any time other than t_{des}

3.4.4 Scenario B

In this scenario Pfister *et al* address the imposition in scenario A, of no post-synaptic activity whatsoever, other the single spike at t_{des}. In place of this rather extreme constraint, scenario B allows for a certain amount of spontaneous post-synaptic activity to be present and the focus is on maximising the firing rate of a single neuron at a desired time t_{des}.

3.4.5 Unconstrained Scenario B (B_u)

The goal here is to maximise the firing rate of the post-synaptic neuron at time t_{des}. There are two differences between this scenario and scenario A_u. Firstly, the stochastic intensity $\rho(t|x, y_t)$ is dependent on the spiking history and so an

additional quantity is introduced to account for spontaneous post-synaptic firing that is independent of spiking history. Secondly, the single pre-synaptic neuron of scenario A is replaced here with 200 pre-synaptic neurons. The output of each of these 200 neurons is a single spike at a time $t_j = j\delta t$, where $\delta t = 1ms$.

The result of this is that the input is essentially a sequential cascade of activity across the whole range of input neurons, consisting 200 spikes in total and is described as $x = \{x_j = \{t_j\}, j = 1, ..., N\}$ where, $N = 200$ in this case. The experimental setup can be seen in figure 3-5 which is taken from Pfister *et al* (2006, p.1329). Each neuron only fires once.

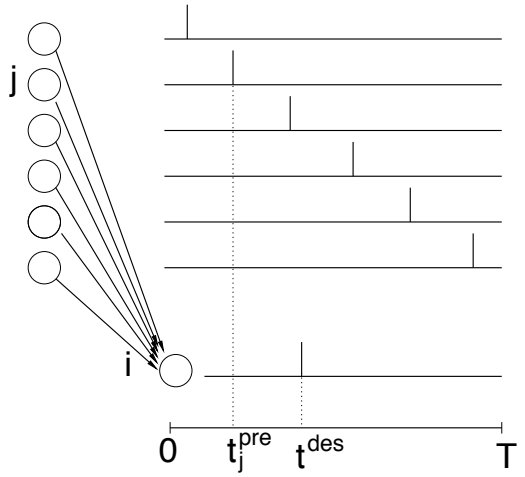


Figure 3-5: Illustrative setup used in the scenario B experiments. Pfister *et al* (2006, p.1329)

3.4.6 Constrained Scenario B (B_c)

The constraint applied in scenario B_c, is analogous to the constraint applied to scenario A_c. It is required that, at times other than t_{des}, the deviations of the instantaneous firing rate be kept to a minimum, with respect to what Pfister *et al* calls a *reference* firing rate ν_0.

Pfister introduces a *penalty* term, P_B in order to minimise the firing rate devia-

tions at times other t_{des}. The form of P_B is as follows:

$$P_B = exp(-\frac{1}{T}\int_0^T \frac{\bar{\rho}(t|x,t_{des})-\nu_0)^2}{2\sigma^2}dt).$$

Therefore, if the parameter σ is small then P_B will be large and conversely, if $\sigma \to \infty$ then the value of P_B will be negligible. Once again, the desired outcome is to determine the optimal synaptic weight update rule in this particular scenario. The optimal solution will be one that gives a high firing rate $\bar{\rho}$ at t_{des} and a spontaneous firing rate at all other times that deviates from ν_0 by as small an amount as possible. The optimality term is given by Pfister *et al* as:

$$L^{B_c} = log(\bar{\rho}(t_{des}|x)P_B)$$

Which then means that the gradient ascent update rule is given by:

$$\Delta w_j^{B_c} - \alpha\frac{\delta\bar{\rho}(t_{des}|x)/\delta w_j}{\bar{\rho}(t_{des}|x)} - \frac{\alpha}{T\sigma^2}\int_0^T(\bar{\rho}(t|x,t_{des}) - \nu_0)\frac{\delta}{\delta w_j}\bar{\rho}(t|x,t_{des})dt.$$

Once again, this weight update rule can be considered to be similar in form to an STDP derived learning function and the plots of $\Delta w_j^{B_c}$ against $\Delta t = t_{pre} - t_{des}$, for both the unconstrained and constrained scenarios can be seen in figure 3-6, taken from Pfister *et al* (2006, p.1331). The plots show the learning curves obtained for different values for the stochasticity — as tuned for by Δu — for the unconstrained scenario and for different values of standard deviation σ in the constrained scenario.

3.4.7 Scenario C

The emphasis here is on pattern recognition. The experimental setup is the most complex of all three scenarios. Consider N pre-synaptic neurons, each of which are connected to each of M post-synaptic neurons as shown in figure 3-7 from Pfister *et al* (2006).

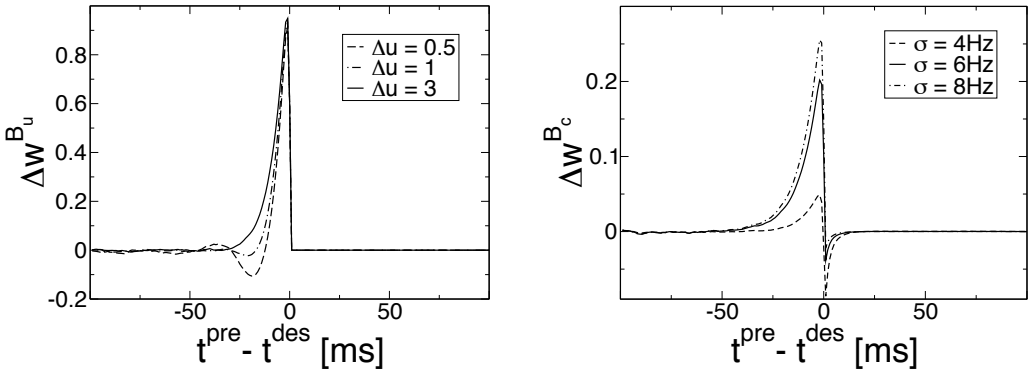


Figure 3-6: Plots of the learning curves resulting from using differing levels of stochasticity, i.e. the width of the threshold region of the neuron is altered — thus altering the stochasticity of the neuron (A), and for differing deviations from the reference firing rate (B). The figure is taken from Pfister *et al* (2006, p.1331).

3.4.8 Pattern Detection — Unconstrained Scenario C (C_u)

The goal now is to train each of the post-synaptic neurons to respond to a simple but precise input pattern from the N pre-synaptic neurons with a specific output spike train, and not to respond at all to other pre-synaptic input patterns. So, in the notation of Pfister *et al*, it is required that a post-synaptic neuron i, when presented with a specific and temporally precise stimulus x^i, will produce a specific and temporally precise output spike train y^i. Additionally, if the post-synaptic neuron is presented with input x^k where $k \neq i$ then no output spike train is produced *i.e* $y^i = 0$. As before the optimality solution must maximise the probability that this is indeed the case. The optimality term for this scenario is ultimately given by Pfister *et al* (2006) as:

$$L^{C_u} = \sum_{i=1}^{M} log(P_i(y^i|x^i)) + \gamma\langle log(P_i(0|x^k))\rangle_{x^k \neq x^i}.$$

Where γ is a measure of how important a particular pattern is i.e. it distinguishes patterns that shouldn't be learned from those patterns that should.

The resulting gradient ascent update rule is then given by:

$$\Delta w_{ij}^{C} = \alpha \tfrac{\delta}{\delta w_{ij}} \log(P_i(y^i|x^i)) + \alpha\gamma\langle \tfrac{\delta}{\delta w_{ij}} \log(P_i(0|x^k))\rangle_{x^k \neq x^i}.$$

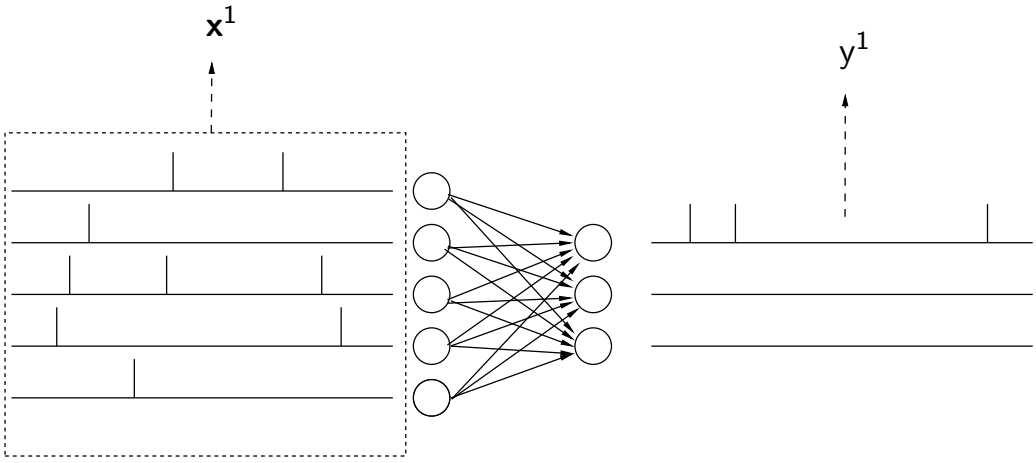


Figure 3-7: Illustrative setup used in the scenario C experiments. Pfister *et al* (2006, p.1332)

The learning rule is applied as a *batch* update, i.e. the update is applied after all input/output patterns have been presented to the post-synaptic neuron. It is noted in Pfister *et al* (2006) that, a fundamental difference between this and the previous scenarios is that whereas before it was possible to plot the update as a function of $\Delta t = t_{pre} - t_{des}$, this is not possible within scenario C as each pre and post-synaptic neuron can emit output spike trains consisting of multiple spikes. Instead Pfister *et al* obtain for the total weight change in terms of a sum of contributions from the STDP function of each pair of pre and post-synaptic spikes, such that:

$$\Delta w_{ij}^{C} = \sum_{t_{pre} \in x_j^i} \sum_{t_{des} \in y^i} \Delta W^{C_u}(t_{pre} - t_{des}).$$

This means that the interaction between each pre-synaptic spike and each post-synaptic spike are considered and summed.

Figure 3-8 is taken from Pfister *et al* (2006, p.1333). There are 400 pre-synaptic neurons and a single post-synaptic neuron. The panels down the left side represent a situation in which the post-synaptic neuron is presented with a pattern that it has been trained to respond to whereas, the right hand panels are for a

situation in which the neuron is presented with a pattern to which it must not respond. The top panels show the input spike trains from each of the 400 pre-synaptic neurons. The middle panels show the result of 1000 spike train responses of the post-synaptic neuron in each situation i.e. to pattern x^i on the left and to a pattern x^k on the right. The bottom panel shows the probability density of firing of the post-synaptic neuron for pattern x^i on the left and x^k on the right.

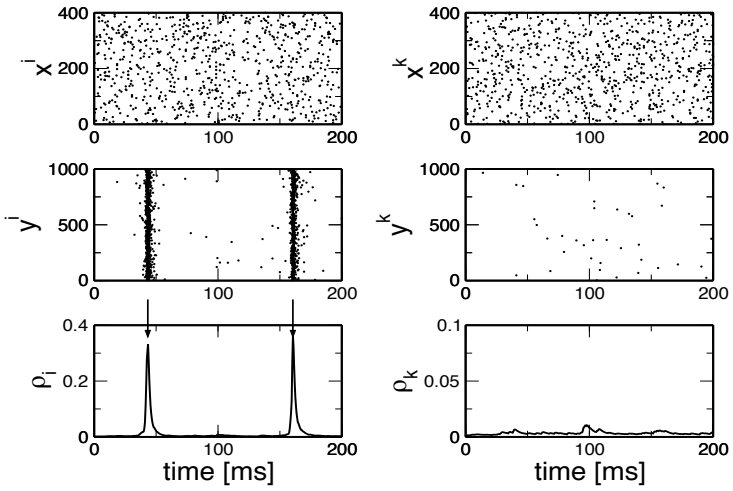


Figure 3-8: Raster plots of Pattern detection results, Pfister *et al* (2006, p.1333).

It can be clearly seen from these results that the use of the update rule does indeed allow for the post-synaptic neuron to respond only to a specific input pattern. The plot of the update function vs $\Delta t = t_{pre} - t_{des}$ can be seen in figure 3-9 from Pfister *et al* (2006). Once again the resulting STDP function has a negative offset that is a product of the constraints that the neuron must only fire in response to a specific pattern and no other, and that it must only produce spikes as specific desired times in response to the appropriate pattern.

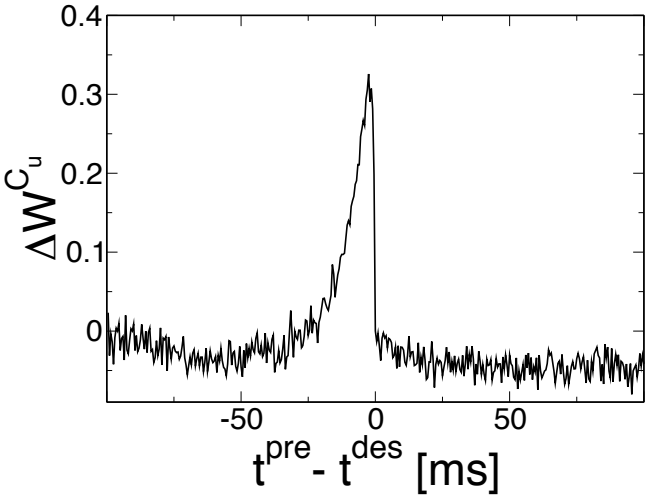


Figure 3-9: Unconstrained scenario C learning curve, Pfister *et al* (2006, p.1335).

3.4.9 Temporal Localisation of STDP learning function — Constrained Scenario C (C_c)

Pfister *et al* (2006), elucidate that the problem with scenario C_u just described is that, the presence of the negative offset of the STDP type learning function implies that the function is not localised in time around $t_{pre} - t_{des} = 0$. Locality of the STDP learning function is a fundamental property that should be satisfied for a system to be realistic. Without locality there can be no realistic modeling of memory because essentially every single pre and post-synaptic spike interaction would be remembered in some form forever.

As a solution, Pfister *et al* modify the update rule in such a way as to penalise increasingly temporally non-local terms. In other words, they increasingly penalise pre and post interactions as $|t_{pre} - t_{des}| \rightarrow \infty$. Pfister *et al* (2006) introduce the variable, a which, allows for the control of the how *local* one wishes the STDP function to be, i.e. how large $|t_{pre} - t_{des}|$ can be before it is penalised. Plots of learning function of different degrees of locality can be seen in figure 3-10, (Pfister *et al*, 2006, p.1335).

Figure 3-11 shows the results obtained for the Optimal STDP function for dif-

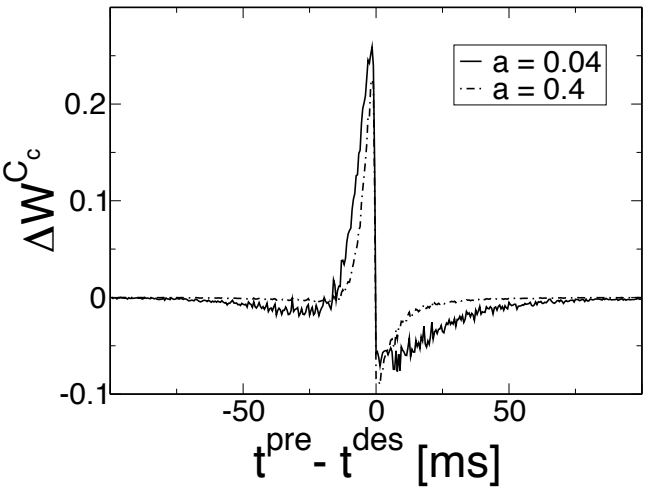


Figure 3-10: Constrained scenario C learning curve, Pfister *et al* (2006, p.1335).

ferent numbers of input patterns with $a = 0.04$, $N = 400$. Another result of this work is shown in figure 3-12 and shows that if the synaptic weights are initialised such that they are small then the resulting STDP function will have a dominating potentiating effect, whereas if the weights are initialised to be large, then the resultant STDP function will have a dominant depressing effect.

3.5 Summary and Conclusions

The principle results of this work are best summarised in figure 3-13, (Pfister *et al*, 2006, p.1340). This figure shows the form of the optimality term in each of the six scenarios just discussed.

Scenario A_u focusses on the most simple requirement of training a post-synaptic neuron to fire first at a desired time t_{des} with no other constraints. The outcome of this is that repeated application of the optimal solution does not result in meaningful, converging synaptic efficacies. If $(t_{pre} - t_{des}) \geq 0$ the weights tend to there is no single synaptic weight that will satisfy optimality and if $(t_{pre} - t_{des}) < 0$ the synaptic efficacy tends to infinity.

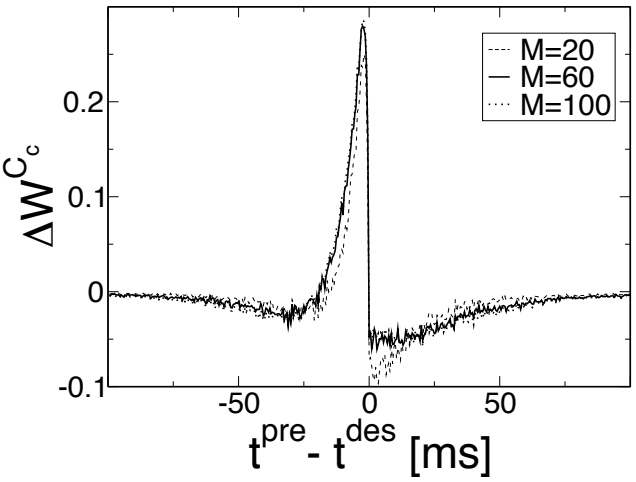


Figure 3-11: Learning curves obtained when the post-synaptic neuron are presented with different numbers of input patterns and are required to respond to only one of those patterns, Pfister *et al* (2006, p.1336).

This is the most simple model, and its short comings are due to the lack of any modeling of synaptic depression.

Scenario A_c is, essentially, a remedy to the lack of depression of model A_u. The requirement is same as for A_u with the addition that, the post-synaptic neuron should not fire at *any* time other than t_{des}.

Scenario B addresses the strict constraint of A which says that there should be no spikes of the post-synaptic neuron whatsoever, other than at t_{des} — perhaps an unrealistic requirement. What scenario B introduces to the model is the ability to have spontaneous post-synaptic spiking that can occur with a realistic frequency ν_0. Scenario B_u is analogous with A_u, the difference being that in B_u Pfister *et al* are attempting to maxmise the firing rate at time t_{des} and not just make the neuron fire a single spike at t_{des}, as in A_u.

Scenario C presents the most interesting experimental setup. In scenario C_u the goal is now to train a post-synaptic neuron to emit multiple precisely timed spikes in response to specific input pattern. The same neuron should not respond to any other input pattern that may be presented to it. The results indicate that

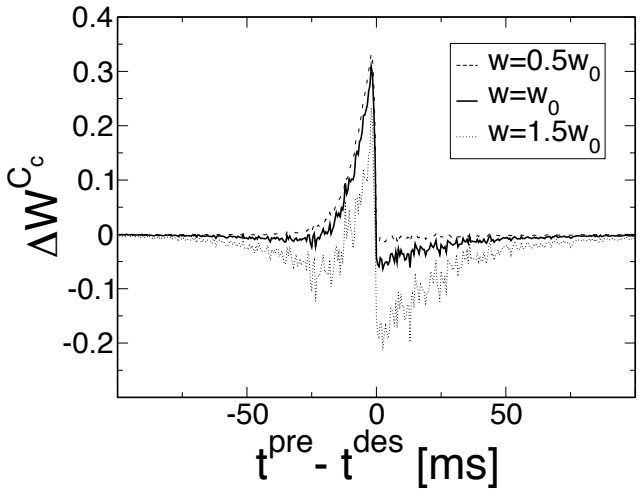


Figure 3-12: Learning curves w.r.t. the size of the initial synaptic weights. Initial weights smaller than w_0 will generally tend to increase, whereas with initial weights larger than w_0 synaptic depression will dominate, Pfister *et al* (2006, p.1336).

such a post-synaptic neuron with 400 input synapses is capable of producing an output spike train of precisely timed spikes in response to a specific input pattern. However, the limitations of the scope of the work here are that the spike train on which the neuron is trained simply consists of two precisely timed bursts of spikes. In terms of complexity, this is a relatively simple spike train. It leaves open the question as to whether or not such a neuron can be trained, via a biologically realistic method, to emit a complex sequence consisting of more than two spikes, in response to a particular input pattern.

The modified scenario C_c addresses the negative offset of the resultant STDP function. As previously stated, the offset implies that the function is not localised to $\Delta t = 0$. The penalisation of increasingly non-local *pre-post* interactions, the strength of which is determined by the parameter a, fixes the locality problem. The resulting localised STDP function compares well to the typical STDP function and, by design, possesses many of the same qualitative features.

This work shows the form to which an STDP learning function should tend to take if it is to be considered 'optimal'. Three scenarios were considered, each differing in terms of the methodology involved in providing the stimulus and the

Unconstrained scenarios		Constrained scenarios	
A_u	*pre-before-post* LTP $\sim$ EPSP	A_c	*post-before-pre* LTD (or LTP) $\sim$ spike afterpot.
B_u	*pre-before-post* LTP/LTD $\sim$ reverse correlation	B_c	*post-before-pre* LTD $\sim$ increased firing rate
C_u	*pre-before-post* LTP $\sim$ EPSP LTD $\sim$ background patterns	C_c	*post-before-pre* LTD $\sim$ background patterns $\sim$ temporal locality

Figure 3-13: Principle results of all constrained/unconstrained scenarios, (Pfister *et al*, 2006, p.1340)

task to be performed. Optimality was defined by the objective function L and synaptic weight changes were made according to a gradient ascent rule performed on L.

The issue of a single neuron learning more complex precise spike trains is covered in Legenstein *et al* (2005) and in Chapter 6 of this dissertation. Additionally, the work in chapter 6 investigates, among other things, the ability of a single neuron to learn — via STDP learning on its input synapses — *multiple* precise Input/Output associations in such a way that all I/O associations can be remembered by the input synaptic weights to the neuron simultaneously — without erasing each other. Also shown in chapter 6 is an example of learning using a non-biologically inspired method to also enable a single neurons to learn to produce precise output spike trains. An outline of the work carried out in Legenstein *et al* (2005), on learning precise spike trains is presented in chapter 3.6 of this dissertation.

3.6 Complex Precise Spike Trains

The work discussed so far in this chapter has dealt with the optimal learning of very simple precise spike trains consisting of only two precisely timed spikes. The research presented in Legenstein *et al* (2005), investigates the ability of a single neuron to learn a far more complex spike train consisting of up to approximately

50 precisely timed spikes, over a 2 second interval. The learning method used by is a form of STDP learning applied to the input synapses of a spiking LIF neuron. During training, the LIF neuron is forced by a teaching input to perform a target spike train in response to the spiking input activity the neuron receives from its input synapses. The connectivity of the experimental setup is essentially the same as that seen in figure 3-1.

The details of the LIF model used along with details of the synaptic modification can be found in Legenstein *et al* (2005). While a large portion of the work in Legenstein *et al* (2005), involves a theoretical analysis, it is the experimental aspect of the work that is of greater relevance to the research in this thesis and it is this experimental work that shall be discussed in greater detail in this introduction.

3.6.1 Uncorrelated Input Experiment

In Legenstein *et al* (2005), a LIF neuron receives 100 inputs from dynamic synapses of which, 90% are excitatory and 10% are inhibitory. The sign of these synapses is fixed, i.e. STDP training cannot cause an inhibitory synapse to become excitatory, and *vice versa* for an excitatory synapse. The target input weight vector was chosen as follows. Half of the excitatory input synapses were assigned their randomly assigned maximum possible value w_{max}, while the remaining half were set to zero. The inhibitory synapses remain fixed for the duration of all experiments performed.

This weight vector is the target weight vector and the threshold of the neuron was set to ensure that when receiving 100 uncorrelated poisson input spike trains each with a frequency of $20Hz$, the LIF neuron would produce an output spike train of frequency $25Hz$. With this target weight vector and adjusted threshold, the neuron is said to perform the transformation F on the 100 input spike trains.

For training, the excitatory synaptic weights were set to a spread of, relatively, very low values (SD and mean can be found in Legenstein *et al* (2005)). The

neuron was then presented with a continuous stream of input from the 100 input synapses for a biologically simulated time of 3600 seconds. These 100 spike trains are uncorrelated and have a frequency of $20Hz$. The neuron also received input from a teaching stimulus which, caused the neuron to fire at times when a neuron with the target input weight vector would have fired. For the duration of this simulation, STDP was performed on the excitatory input synapses, according to the STDP rules set out in Legenstein *et al* (2005).

The results of this experiment show that over the course of the learning, the correlation between the actual output of the neuron and the target output, increases significantly — while error decreases. This is shown in figure 3-14, (Legenstein *et al*, 2005, p.2364).

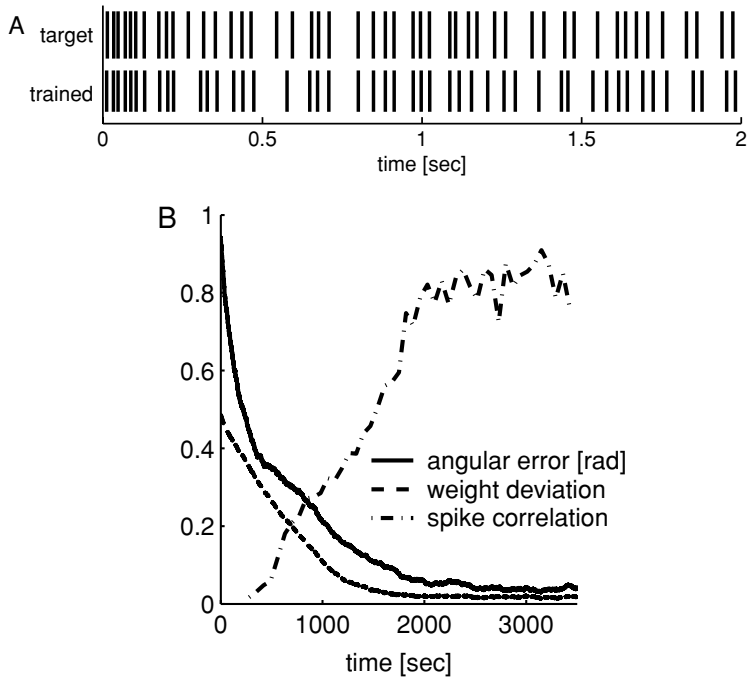


Figure 3-14: The top figure contains a target spike train and the actual spike train of the trained spiking LIF neuron. The lower figure clearly illustrates, in a quantitative manner, the decreasing error and deviation between the target and actual spike trains, along with their increasing correlation as exposure to STDP training increases, (Legenstein *et al*, 2005, p.2364).

Legenstein *et al* replace each spike in the target and actual spike trains with a Gaussian function with an SD of $5ms$. The correlation is then given by the correlation of these two functions of time, and was calculated over $100ms$ segments. This result was also shown to hold, even if the number of inputs to the neuron varied — from 25 to 200. These results are shown in figure 3-15.

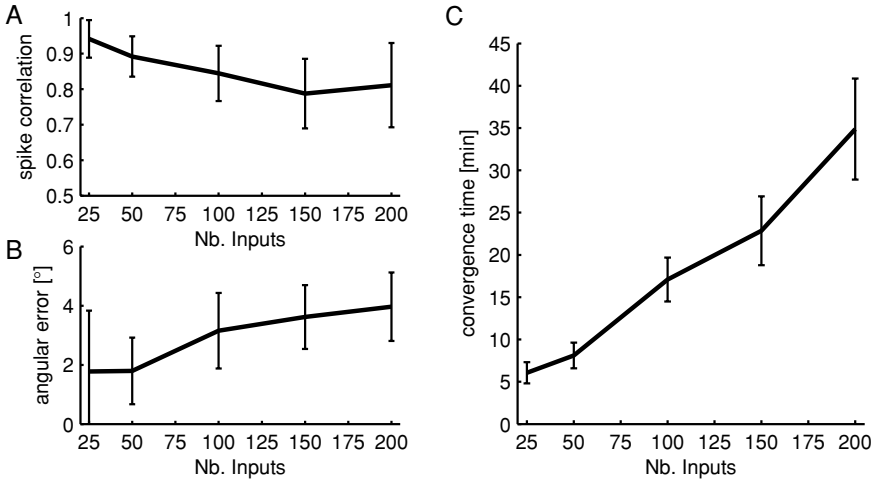


Figure 3-15: The two graphs on the left show that the spike correlation and error between target and actual spike trains holds for different numbers of inputs. The right-hand graph clearly shows that the time to convergence increases in an approximately linear manner, as the number of inputs to the LIF neuron increases, (Legenstein *et al*, 2005, p.2366).

This experiment was also repeated with a 'noisy' teacher stimulus. An element of noise was introduced to the teaching stimulus by means of 'jittering' the timing of the spikes of the stimulus with Gaussian noise — zero mean and SD $4ms$. It was found that while the result was essentially the same, the training time required to get this result was greatly increased over the noiseless teaching stimulus scenario.

3.6.2 Correlated Input Experiment

Using a similar experimental setup to that considered above, with the 90 excitatory inputs being divided into 9 groups of 10 synapses. The input spike trains

were highly correlated within each group, while there is hardly any correlation between groups, (Legenstein *et al*, 2005). The method of correlation used by Legenstein *et al*, is taken from Gütig *et al* (2003). The target transformations to be learnt are those that require different weights for highly correlated spike trains. The actual form this takes is as follows: 5 of the synapses of each of the 9 groups would have their weight set to zero and the other 5 in each group would have their weight set to their randomly chosen maximal value. The learning takes place in the presence of non-teacher-induced firing, by which Legenstein *et al* mean; firing of the neuron that was not due to the input from the teaching stimulus. While it was found that learning was somewhat successful, it was also found that it was better if the neuron received additional inhibitory inputs to counter the effect of the neuron firing in response to highly correlated input spikes. This inhibition was in the form of 30 inhibitory inputs with uncorrelated spike trains. As a result the neuron was more likely to fire during training, in response only to the teaching stimulus. The target weight vector and the learnt weight vector can be seen in figure 3-16. Full results for this experiment can be found in Legenstein *et al* (2005).

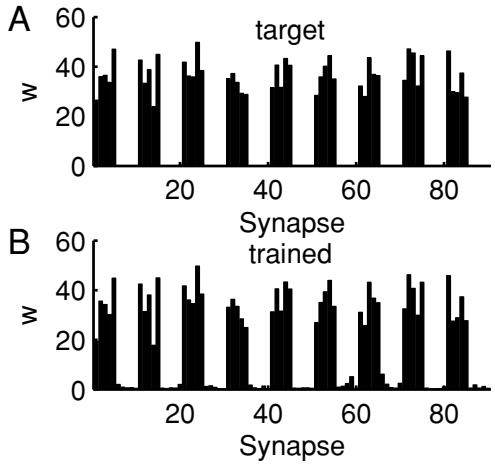


Figure 3-16: The top panel shows the relative target values of the synaptic weights, while the lower panel shows the actual synaptic weight values that have been learned. Taken from Legenstein *et al* (2005, p.2368).

3.6.3 Dependence of Learning Performance on Input Correlation

Consider the same experimental setup just described in section 3.6.2. The focus of this experiment is to investigate how correlation among the input spike trains can affect learning. The correlation between the inputs for this experiment are as follows: the spike trains of 4 of the input groups — with each group consisting of 10 inputs — are constructed such that the correlations between the spike trains of inputs within each group are given the same value. The remaining 50 input spike trains, and the 10 inhibitory inputs were uncorrelated. The neuron also receives the additional inhibitory input from 30 synapses as in the previous experiment.

The correlation measure was also more pronounced — or sharpened — in this experiment, as the time-constant of the Gaussian function used to determine the correlation functions, was reduced from $10ms$ to $6ms$. The neuron was required to learn the target transform described above for the correlated input experiment and the results, over 20 trials, can be seen in figure 3-17.

It can be seen from figure 3-17, increasing the correlation between the input spike trains, results in a decreasing correlation between the actual output of the neuron and the desired output. It is noted by Legenstein *et al* that, for even the highest levels of input correlation, the correlation between the actual and the desired output of the neuron after training is still not too bad but, the weight vector at this point is not close to the target weight vector. So it appears that, for high input correlations, many weight vectors can produce similar output spike trains. To demonstrate this, figure 3-17 also shows a plot in which the weight vectors are assigned randomly — the dotted line. Where the input correlation is high, then even with the randomly assigned weight vector, the neuron can still produce an output that is of a similar correlation to the desired output as in the case of the neuron with the trained weight vector (solid line), at least in terms of the correlation measure used by Legenstein *et al* (2005).

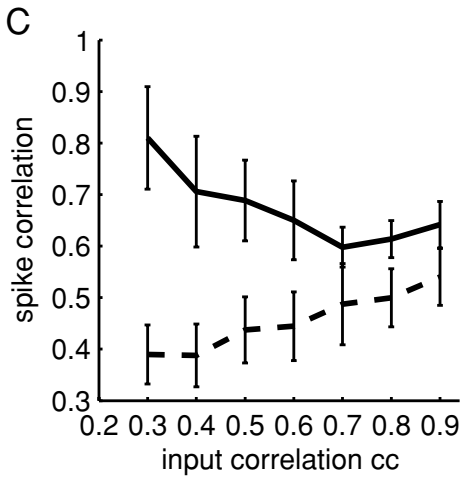


Figure 3-17: Plot of the spike correlation function for the LIF neuron (y-axis) against the correlation strength of the inputs (x-axis). The plot shows the results over 20 runs for 7 particular input correlation values that were chosen, where each run was for a simulated duration of 1 hour (solid line). The dotted line shows the result of randomly assigning a weight value of zero or the maximum allowed value, to *all* synapses.Taken from Legenstein *et al* (2005, p.2368).

3.6.4 Time-varying Input Correlations

It was also found by Legenstein *et al*, that using a form of time-varying correlation between the input spike trains also produced good learning results. The nature of the time-varying correlation is based on work by Song *et al* (2000). The cross-correlation function obtained, decays exponentially with a time constant τ_c. All results and parameters can be found in Legenstein *et al* (2005).

3.6.5 Modification of STDP Rules

STDP learning functions are typically applied to all pairs of pre and post-synaptic spikes. A different approach was considered in Froemke and Dan (2002). In this alternative treatment of STDP, the plasticity arises not from the repetition of pairings of pre an post-synaptic spikes but instead, from the use of much longer pre and post-synaptic spike trains and in which the efficacy of a spike from a

neuron depends on the time since that same neuron last emitted a spike, such that in the case of two spikes from the same neuron being temporally close, the effect of the later spike on any post-synaptic neuron, is reduced according to $\epsilon_i = 1 - exp(-(t_i - t_{i-1})/\tau_S)$, where t_i and t_{i-1} are the times of the i^{th} and the $(i-1)^{th}$ spike respectively and where τ_S is the time constant — referred to by Froemke and Dan as the spike suppression constant.

Using this modified version of STDP, Legenstein *et al* found that, compared to the unmodified version of STDP they had previously used, learning performance was reduced in the case where the correlation between input spike trains was high, and that there was no effect — or perhaps a small positive effect — on learning performance for other input correlations. A table of correlations for the modified and unmodified versions of STDP can be seen Legenstein *et al* (2005, p.2371).

A form of STDP was also considered by Legenstein *et al* which, allows the synaptic weights to assume stable values between 0 and their maximum value w_{max}. Using this form of STDP, it was found that the learning of target transformations that require stable intermediate synaptic weights was possible but that, the resulting correlation measure between actual and desired output spike trains was not as good as for those target transformations that only required synaptic weights to assume either their minimum or maximum values. It is noted by Legenstein *et al* that successful learning was highly sensitive to the values of the learning parameters which control the form of the STDP function.

STDP implementations which are capable of producing stable, intermediate synaptic weight values are desirable as a learning method, especially over more basic STDP implementations, due to the fact that their ability to allow the synaptic weights to assume significantly more diverse values than just the minimum or maximum, means that they are capable of learning a larger number of target transformations than the more basic STDP implementations, (Legenstein *et al*, 2005).

3.7 Recapitulation and Summary

This chapter of the thesis has introduced some of the prior research on precise spike trains most relevant to the original work contained in this dissertation. The research of Bi and Poo (1998, 1999, 2001), is generally considered to be the most relevant in terms of recent study into the biological mechanisms. This work by Bi and Poo is used as the basis for many artificial STDP learning mechanisms such as those considered in this dissertation, Pfister *et al* (2006), Legenstein *et al* (2005), and some aspects of the work contained in Gerstner and Kistler (2002), among many others. Bi and Poo (1998), provides real experimental confirmation, using samples of real neurons, that correlations of pre and post-synaptic firing, causes an enhancement of the efficacies of the correlated inputs, the strength of which is dependent on the duration of time between pairs of pre and post-synaptic spikes.

Legenstein *et al* (2005) demonstrates that a form of STDP learning can increase the correlation between the output spike train of a post-synaptic LIF neuron and some target precise spike train — as measured by their previously discussed spike correlation function. It was shown that this result also holds for: *i*) Increasing network size — from 25 to 200 input neurons; *ii*) Using a 'noisy' input. However, the introduction of noise did cause the training time to increase significantly when compared to the noiseless case.

Additionally, Legenstein *et al* also found that it was possible, using their form of STDP learning, for input synapses to the post-synaptic neuron to be trained to learn highly dissimilar weight values, even when their input spike trains are highly correlated. This is a hard learning scenario as it requires connections that are very similar in terms of their input spike trains, to achieve weight values that are as different as zero and the maximum weight value. The effect of correlation between input spike trains to the post-synaptic neuron is also investigated. Legenstein *et al* showed that at higher input spike train correlations, not only do different input weight vectors cause the post-synaptic neuron to produce similar output spike trains, but that also these higher input correlations reduce the ability for STDP

learning to enable the post-synaptic neuron to produce an output spike train that is highly similar to the target spike train, and that, in fact, the actual output to target output correlation at these high input correlations is statistically similar to the spike correlation of a post-synaptic neuron with a randomly assigned weight vector.

Furthermore, an implementation of STDP is considered in Legenstein *et al* (2005) which, allows the synaptic weights to assume intermediate values that lie between zero and the maximum. It was found that such an implementation produces stable weight vectors and this learning method can learn a greater number of target transformations than the scenario in which weights are restricted to minimum and maximum values. The disadvantage was found to be that the spike correlations between the target and the actual output is not as high as in the weight restricted case, Legenstein *et al* (2005).

Part II

Original Work

Introduction

The research field and the current state of the research in this field has now been established. The LSM has been introduced along with numerous investigations of its computational capabilities. Additionally, examples of the recent research that has been performed on the investigation of learning precise spike trains using STDP inspired learning techniques has also been reviewed. This next section presents the original work that forms the core of the thesis.

In Chapter 4 the emphasis of the research is on applying spiking neurons and structures built out of spiking neurons, in a manner that is very different to the typical applications that have been discussed in part I. The applications discussed in this chapter for networks of spiking neurons could possibly be implemented in hardware to introduce a novel method of computing.

Chapter 5 introduces a learning regime based on Hebbian learning, Spike Time Dependent Plasticity (STDP) learning with normalising ($STDP + N$). In itself, this is not a particularly new learning regime. However, what this chapter presents is an analysis of some of the fundamental aspects of this learning regime, when applied to synapses within a recurrently connected network based on an LSM. An expression is formulated, the activity link vector L, which relates the input and output spiking activity of a spiking LIF neuron to the resultant weight change. Analyses are performed that investigate the effect of Hebbian followed by Anti-Hebbian $STDP + N$ learning, on the modification of network synaptic weights.

The work of Pfister *et al*, examining the capability of individual LIF neurons to learn temporally precise patterns has been introduced and discussed. This work is related the original work in chapter 6. The work in chapter 6 expands on the work investigating the ability of an individual neuron to learn a temporally precise firing pattern, some of which was performed concurrently with that of Pfister *et al* (2006), Toyoizumi *et al* (2007) and Legenstein *et al* (2005), and investigates the ability of a single neuron to learn *more than one* of these patterns simultaneously. This chapter also introduces a metric for weighted time series of spikes that is

used to determine how well the single neuron has learnt a goal precise spike train response to a collection of specific input spike trains.

Chapter 7 investigates the method of application of $STDP$ learning within an LSM-style network. In the work discussed in part I, in particular the work seen in Hæusler and Maass (2007), the learning is applied to all synapses of the recurrent network. Chapter 7 provides an investigation into a novel application of the learning regime within a recurrent network, with a view to improving the learning according to observations of the sequential firing intervals that groups of network neurons are capable of learning. A form of STDP learning is introduced and it is demonstrated that this form of learning can enable networks of recurrently connected spiking neurons to learn firing patterns that the standard application methods of STDP learning would not be capable of.

Chapter 4

Parallel Computation in Spiking Neural Nets

4.1 Introduction

A great deal of work is currently going on to design and build spiking neural nets as nanostructures in silicon. See, for example, the recent paper by Bindal and Hamedi-Hagh (2007). A review of a variety of different neural models, along with feasible techniques for implementing these models in hardware, can be seen in Smith (2006). In connection with this, and the ongoing efforts to understand the computational powers of biological neurons, as reviewed, for example, in the paper by Herz et al (2006), it seems important to understand the potential computational power of artificial spiking neural networks.

Siegelman (1999), shows that the family of analog neural nets with saturated sigmoid transition function, considered as computing devices, are Turing complete. This means that any computation which can be carried out by a Turing machine with its potentially unbounded memory can also, in principle, be carried out by finite networks of these analog devices. This is possible because the neurons hold real numbers, each of which may contain an unbounded amount of information.

The essence of Siegelmann's construction is to consider a real number as a stack of binary values, via the usual binary decimal expansion. Neto *et al* (2003) have also shown that a general purpose parallel programming language, similar to Occam, (Inmos Limited, 1998), can be compiled into one of these sigmoid neural nets.

The case of spiking neural nets is somewhat different, because here the messages sent between neurons are series of stereotyped spikes. The only information carried by a spike is the time at which it occurs. We may, it is true, look at a spiking network over a long period of time and record the rate of firing, averaged over some time window, of each neuron at each instant. This rate coding transformation from one model to the other is not entirely convincing, since the number of possible values represented by a rate would depend on the size of the time window, but nevertheless it does suggest that spiking neural nets ought also to have the property of Turing completeness. This was actually shown by Maass *et al* (1996). In this case real numbers, within a certain interval, are represented as phase differences of oscillating systems. That is, the information is held in the dynamics of the system, not directly as in the work of Siegelmann. As before, however, the essential idea is to use the real numbers as binary stacks.

Both of these constructions give more than Turing completeness. They actually implement the real number machines of Blum *et al* (1997). A real number machine is one in which the memory registers can contain real numbers and the machine is able to perform computations using the reals whereas, the memory registers of a turing machine do not contain real numbers, only characters from some finite sized alphabet are allowed. Of course the implementations are in neural nets considered as dynamical systems, as mathematical abstractions, and not in actual networks of real neurons.

The Maass construction also depends on some special assumptions about the postsynaptic response function, the most important one being that it is linear on some interval. The results of this chapter are closely related but the methods are different

The motivation for the work within this chapter is that it utilises neural networks in a different way and for a different purpose from what is, and indeed has been, the norm. Our main contribution is a way to represent integer variables. The natural number n is represented by taking two in-phase copies of oscillating subsystems and performing n standardised perturbations of one of them. To read the number back, the subsystem which was not perturbed is perturbed some number of times until the two subsystems are again in phase, as they were at the beginning. To support this idea, we require some device, made out of spiking neural nets, to detect whether or not two oscillators are in phase. This is called a coincidence detector. In contrast to Blum *et al*, we do not assume unbounded precision accuracy of the coincidence detector. Therefore, we can only compute with integer values of a bounded size, as with ordinary computers. It is shown that a weak version of Occam can be implemented in this way. As an example, it is shown how addition and multiplication could be implemented, albeit inefficiently. We do not need the assumption of linearity in an interval of the post synaptic response function, as used by Maass. We do assume that the post synaptic response function is continuous, that it starts from a resting value, increases to a maximum and then decreases back to the resting value. The need for highly constructed and controlled networks of neurons to perform the basic arithmetic operations seen in this chapter is an indication that such coherent and arithmetical thought may actually be quite a difficult process for biological neural networks to successfully accomplish.

4.2 Oscillations and Spiking Neurons

The spiking neurons and spiking neural networks discussed in this article should be understood as members of the family of deterministic mathematical models, such as leaky integrate and fire, or the spike response model, discussed and described in Gerstner and Kistler (2002).

Numerical quantities can be represented as phase differences between equiperiodic oscillating subsystems in a spiking neural net. It is then possible to repre-

sent integer variables, and the increment and decrement operations, $X := X + 1$, $X := X - 1$. It is also possible to represent some basic parallel programming constructions: if, while, seq, par, alt, as used in, for example, the Occam programming language. However, we do not claim that computations of this kind are actually done in this way in biological systems — we merely propose a novel application for such neurons.

4.3 Terminology

We will say that a network of spiking neurons is an oscillator if it has some initial state so that, without any further input, every neuron in the network will fire periodically, possibly with different periods. It is quite possible for an oscillator to have other behaviours, for other initial states, even without any input. An oscillating state is only determined by a network and a designated initial state. We will say that an oscillating orbit is attractive if any initial state sufficiently near to the designated initial state results in an orbit which spirals in to the oscillating orbit.

Attractive oscillators are very common in spiking neural networks (Wang, 2000) For example, a single neuron with an excitatory link to itself and a time delay will oscillate, if the weight is large enough, and the oscillation is attractive, with rapid convergence, for most values of the parameters. Another example of this type would be a chorus, which is a symmetric network of identical neurons, totally connected by excitatory links, all with the same weights and time delays. For discussion, see Gerstner and Kistler (2002). Such oscillators can be started from the zero state by giving an initial excitatory impulse. The periods can be adjusted or altered, to some extent, by altering the weights and the time delays.

We will say two oscillating subsystems are matched if there is a one to one correspondence between the neurons and links of the subsystems, preserving all the parameters of the neurons and the links.

Two matched oscillating subsystems which are started together will be in phase

forever. If one is started later than the other, the phase difference will be preserved forever. It is these phase differences which we will use as elements of memory. We note that in case there is noise on the links, the phase difference will drift at random over time. To some extent, such random drift can be slowed by duplication of systems.

In the following, we will advance the phase of an oscillating system by sending an exciting impulse to it. This depends on the idea that if a neuron is about to fire in the near future, a small exciting impulse will cause the firing to occur earlier. This in turn depends on some assumptions about the post synaptic response. We assume, in fact, that this post synaptic response is continuous, that it starts from a resting state, increases to a maximum, and then decreases back to the resting state. (These assumptions are simpler and somewhat different from those used in Maass *et al* (1996), where strict continuity is not needed, but some intervals of linearity are required). The continuity is important for our construction, since it allows fine tuning of the response. We consider that this is a realistic assumption. This can be justified if one considers that in reality, the response functions of the post synaptic response of real neurons must also be continuous, no discontinuous jumps in the function can happen in reality. Note that some event based simulation systems actually depend on the contrary assumption of an instantaneous and discontinuous initial jump of the response function in order to simplify their computations. See Delorme *et al* (1999), Mattia and Del Giudice (2000).

In the section below we define a simple concurrent programming language. A program in such a language is just a statement. These statements will be computed in spiking neural nets.

A subsystem is a network of spiking neurons. We will assume that each subsystem has an initiating neuron and a terminating neuron. The subsystem is started by sending an exciting impulse to the initiating neuron. The subsystem finishes, if ever, at the first instant after it starts when the terminating neuron fires. The activity between the start and the finish is called a process. Of course the process depends on the initial state of the subsystem. We do not always assume

that the initial state of a subsystem is zero. We intend to use these processes to implement statements in the programming language. Of course observing the process should tell us all we need to know about the computation which is supposed to be described by the original statement. At this point we will just make the minimum assumption, which is that the process should terminate, given some initial conditions, if and only if the statement in its usual denotational semantics describes a terminating computation. In case of termination, we also expect the terminating state of the variables (described below) to be correct, according to the denotational semantics, provided that none of the integral values which occur in the computation are too large, or too small.

4.4 A Simple Parallel Programming Language

The language described below is a very weak version of Occam, without channels, and only integer variables, and all variables global. See Inmos Limited (1998).

We will use strings of characters starting with upper case letters as variables for integers.

The language consists of statements:

- If X is a variable, then

  $X := 0$, $X := X + 1$, and $X := X - 1$ are statements.

- If X and Y are variables, then

  $X := Y$ and $X := -Y$ are statements. These are all called assignment statements.

- If X is a variable, then

  $X = 0$ and $X \neq 0$ and $X > 0$ are tests.

- If T is a test and P and Q are statements, then

  If T then P else Q is a statement.

- If T is a test and P is a statement, then

  While (T) P is a statement.

We will say that two statements are independent if no variable which may be modified in one statement is referred to in the other.

- If $P_1, \ldots, P_n$ are statements, then

  $Seq\{P_1; P_2; \ldots P_n;\}$ is a statement.

- If $P_1, \ldots, P_n$ are independent statements, then

  $Par\{P_1; P_2; \ldots P_n;\}$, and $Alt\{P_1; P_2; \ldots P_n;\}$ are statements.

Statements in this language have a natural denotational semantics, defined as follows. We define a valuation to be an assignment of integral values to some subset of the variables. Then each statement denotes a possibly non-deterministic, possibly non-terminating, transformation from one valuation to another. These denotations can be defined by structural recursion on the statements, as usual. So, for example, the transformation denoted by $Par\{P_1; P_2; ...P_n;\}$ is obtained by running the transformations denoted by $P_1, P_2, \ldots, P_n$ in parallel, and it only terminates when and if all of them terminate. On the other hand, the transformation denoted by $Alt\{P_1; P_2; \ldots P_n;\}$ is obtained by running all of the denoted transformations in parallel, and it terminates just when one of its constituents terminates, the choice being non deterministic if several of them terminate. See Hoare (1995) for discussion of nondeterministic choice.

We show below how the transformations denoted by statements in this programming language can be implemented in spiking neural networks.

4.5 Representation of Variables for Integers

We define a switch to be a pair of oscillators, each inhibiting the other. So if (A, B) is a switch, it has at least three steady states: A oscillating and B quiet;

both A and B quiet; A quiet and B oscillating. There should be a large number of time delayed inhibiting links between A and B to prevent them from both oscillating together. The simplest case of a switch would be a pair of neurons, A and B, each neuron exciting itself, and the pair connected by many time-delayed inhibiting links. A switch can have one pole or the other turned on, or both turned off. We note that within the context of the deterministic model, there is no fading memory; whatever state a switch is in will persist until further input is given.

We define a variable X to be a switch, (A_0, B_0) together with an array of matched oscillators $(A_1, B_1), \ldots, (A_n, B_n)$. For each i, the oscillators A_i and B_i are matched, and may oscillate independently, but the oscillators $A_1, \ldots, A_n$ are all distinct, with different periods.

Our convention will be that when the value of a variable is positive, the A_0 side of the switch will be on; and when the value of the variable is negative, the B_0 side of the switch will be on.

We will say that a variable $(A_0, B_0), (A_1, B_1), \ldots, (A_n, B_n)$ is exactly zero if (A_0, B_0) are both off, and if, for each $i > 0$, A_i and B_i are in phase.

Very small phase differences may not be observable. Let us agree that we can observe whether or not a phase difference is below ϵ for some $\epsilon > 0$.

We will say that a variable $(A_0, B_0), (A_1, B_1), \ldots, (A_n, B_n)$ is observably zero, with tolerance ϵ, if (A_0, B_0) are both off, and if, for all $i > 0$, A_i and B_i have phase difference below ϵ.

A variable is observably non zero with tolerence ϵ if it not observably zero with tolerence ϵ.

A subsystem which decides, with high probability, whether or not a variable is observably zero (with some tolerence ϵ) will be called a coincidence detector. As mentioned in Gerstner and Kistler (2002), there are actual biological systems of this type, with quite small values of ϵ, in the barn owl for example. There may well be just as precise structures in other species.

Let X be a variable, represented by $(A_0, B_0), (A_1, B_1), \ldots, (A_n, B_n)$. As mentioned above, we use the convention that when X is positive A_0 will be on, and when X is negative B_0 will be on, and when X is zero both A_0 and B_0 will be off, i.e. not firing. We will call $A_1, \ldots, A_n$ the positive part of the variable, and $B_1, \ldots, B_n$ the negative part of the variable. Incrementing the variable will mean incrementing all the positive parts, and decrementing the variable will mean incrementing all the negative parts, and then modifying the switch if necessary. One oscillator, A_i, will be incremented by sending it one exciting impulse, thereby advancing its phase.

It follows that, in order to ensure that each increment has the same effect, input spikes must be 'injected' into each neuron of the oscillator at the same point in the period of the neuron. Obviously another structure is required to ensure that this is the case. We shall call this structure the synchroniser. The task of the synchroniser is to ensure that the effect of an increment is independent of the time when the increment is done. A synchroniser is described below for the simple case in which the oscillator is just one neuron connected to itself. The idea is to hold the incrementing impulse and to deliver it, with a small time delay, when the neuron itself fires.

The natural number n is represented by incrementing zero n times, and resetting the switch to indicate that n is positive. In general, after an increment or decrement, the switch may need to be reset appropriately, if n changes sign.

Let (A, B) be the simplest matched pair of oscillators, each a single neuron self exciting. Let the weight for this exciting link be w, and the time delay d. The period τ of the spiking depends on w, decreasing as w increases. Suppose an increment advances the phase of A by δ. We must have $\delta > \epsilon$, since the result of one increment must observably change the phase. Assume $\delta > 2\epsilon$. After some number $N(\epsilon, \tau)$ of increments the matched pair will again appear in phase, with tolerence ϵ. After about τ/δ increments there is probability of about $2\epsilon/\delta$ of appearing to be in phase. So, roughly speaking, we could expect $N(\epsilon, \tau)$ to be about $\tau/(2\epsilon)$. That is the expected memory capacity of one matched pair. Since all the matched pairs in a variable have different parameters, and

therefore different periods, we expect that the memory capacity of an array of matched pairs would be approximately the product of the memory capacities of the individual pairs.

Let X and Y be variables. The statement $X := Y$ could be implemented by a process which, when initiated, would first turn off all the oscillators of X, and then take input from all the oscillators of Y and pass it on, with the same time intervals, to the corresponding oscillators of X, and would then terminate. We will consider that the subsystem implementing $X := Y$ contains the neurons of X, and takes input from the neurons of Y, but does not contain them. So the subsystems implementing $X := Y$, and $Z := Y$ could be disjoint, even though both take input from the neurons of Y. So we would say that the independent statements $X := Y$ and $Z := Y$ could be done in parallel by disjoint subsystems.

4.6 Demonstrations in CSIM

CSIM (neural Circuit SIMulator), (IGI Group, 2008), is a neural network simulation tool that allows the simulation of networks of different models of neurons and synapses. It has a multitude of user modifiable properties for both synapses and neurons. These range from models such as the basic integrate and fire neuron, to more sophisticated models involving ion channel modeling. Similarly, synapses can be simple static spiking synapses or dynamic spiking synapses which allow the effects of synaptic plasticity to be accounted for.

All CSIM scripts are written and executed in Matlab. It has been used extensively in work by Maass and is designed and written by the LSM Group of the Institute for Theoretical Computer Science at the University of Graz.

We have used CSIM to build examples of the structures described in this chapter.

More detailed information concerning the usage and capabilities of CSIM can be found in IGI Group (2008).

4.7 Single neuron oscillator

First it was necessary to show that it is possible to create a stable oscillator which will oscillate with constant period forever if stimulated and then left undisturbed. The structure of a single neuron oscillator can be seen in figure 4-1 below.

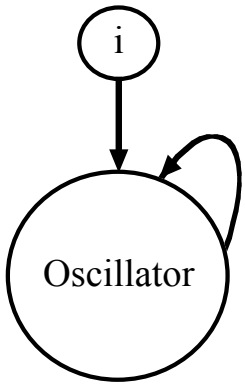


Figure 4-1: A Single neuron oscillator with spiking input neuron i.

4.7.1 Results for a single neuron oscillator in CSIM

Figure 4-2 shows the spiking output of a single neuron oscillator created with CSIM, using a leaky integrate and fire neuron.

4.8 The synchroniser

Consider a matched pair of oscillators, (A, B), as described earlier. A single input spike of sufficient magnitude to one of the oscillators would cause a phase advancement of that oscillator, after which the previous spiking rate is quickly resumed. However, there now exists a permanent phase difference between A and B.

The function of a synchroniser is to ensure that the introduction into an oscillator of a phase advancement spike will only ever occur at or very near to a specific

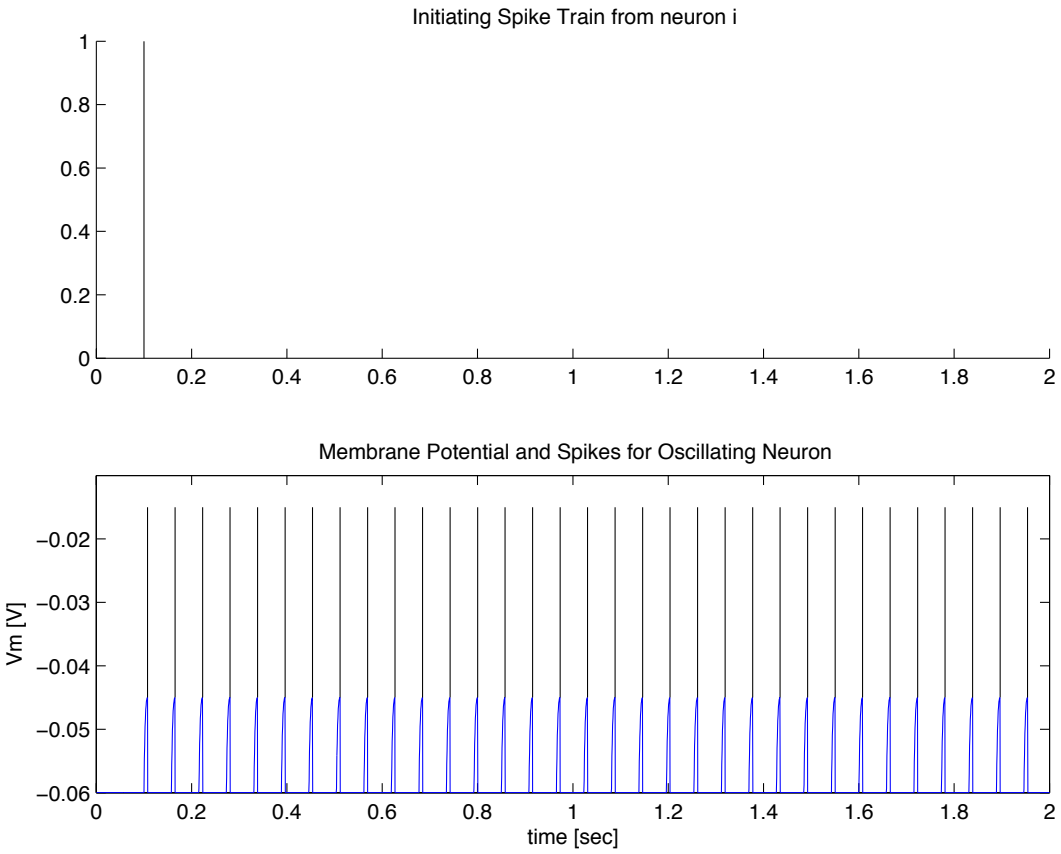


Figure 4-2: The top panel shows the single input spike from input neuron i. The lower panel shows the periodic, oscillating spiking output of the oscillator.

point in the period of the oscillator. This ensures that every input into the oscillator facilitates a near-identical response — preserving the predictability of the system.

If we were allowed to introduce a phase advancement spike at any instant during the period of an oscillator then the instant of spiking of that oscillator would differ for each phase advancement spike. See figure 4-3 below — where the third input spike was ignored completely as it arrived while the oscillator was spiking and was consequently swamped.

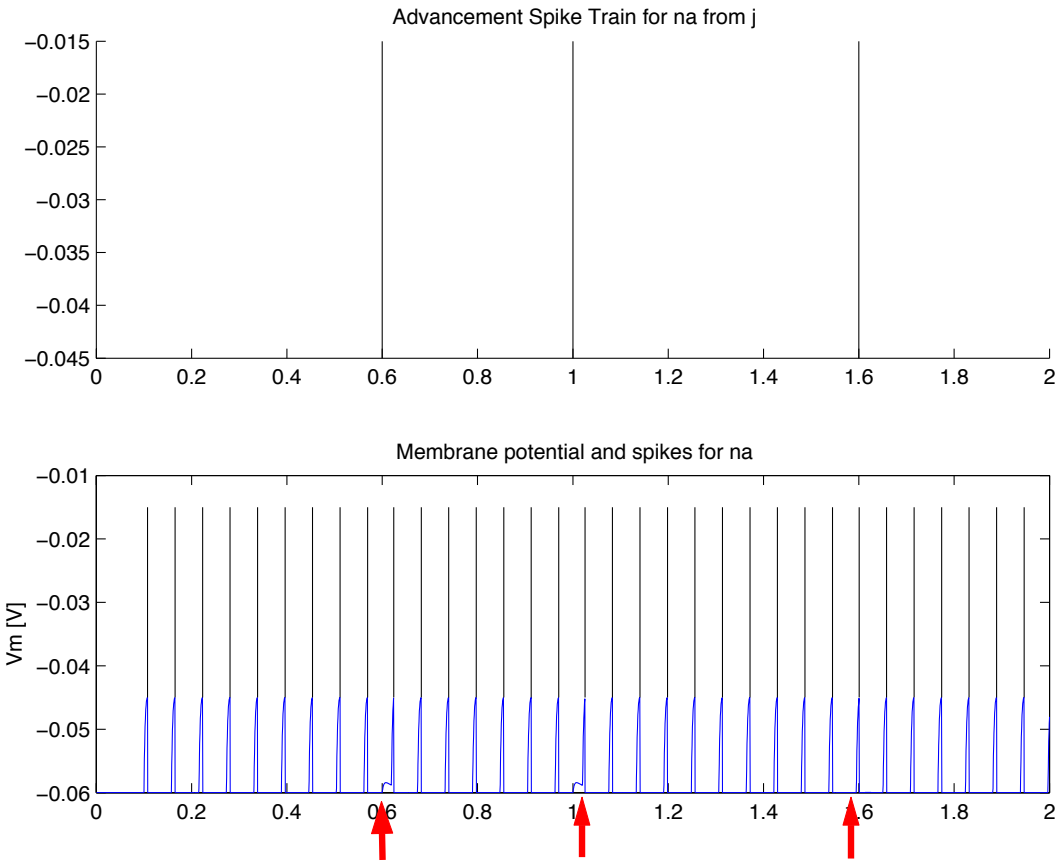


Figure 4-3: Spike injection without a synchroniser. The top panel shows a series of three input spikes from the spiking input neuron i. In the lower panel it can be seen that while the first two spike promote similar responses from the oscillator neuron, n_a, the third input spike produces a different response.

4.8.1 Structure of a synchroniser

Figure 4-4 illustrates a possible design for the synchroniser. This design is the one used throughout the chapter.

The input neuron i emits a spike which will produce a phase advancement of an oscillator A. This spike is relayed to the oscillator via a combination of two structures, the first of which is a switch, the second is known as the accumulator. A precondition for the correct functioning of the synchroniser is that the switch

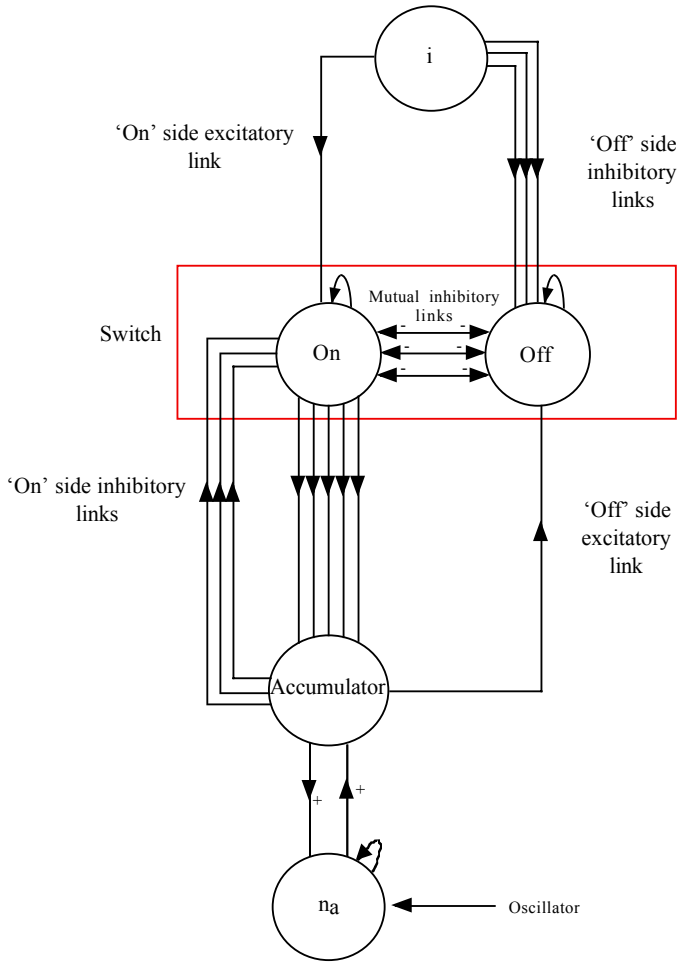


Figure 4-4: The structure of the synchroniser. This structure can be seen to work in figure 4-7.

has been in the 'off' state for a certain period of time before the spiking of the input neuron.

4.8.2 Elements of a synchroniser: The switch

The function of the switch is to indicate the presence of an input spike destined for the oscillator. The 'on' side of the switch has a single, time-delayed excitatory connection from the input neuron i, the weight of which is sufficient to cause

spiking, and it has a connection to itself so that the spiking continues periodically. On the other hand, the 'off' side receives multiple inhibitory time delayed connections from the 'on' side — each of which have the same weight but where the modulus of one of these weights is much smaller than the weight of the excitatory connection to the 'on' side from the initiating neuron. The effect of these multiple inhibitory connections is to temporarily lower the internal state of the 'off' side oscillator by a certain amount so that it is unable to continue to spike.

When the accumulator neuron eventually fires, the switch will also receive multiple inputs from the accumulator neuron, with the 'on' side of the switch receiving multiple time delayed inhibitory connections and the 'off' side receiving a single, time-delayed excitatory connection, reversing the switch.

Once the 'on' side of the switch has been stimulated by i it will spike regularly and continually until the accumulator neuron spikes and sends it multiple inhibitory spikes to shut it down.

There are also mutual inhibitory links between both sides of the switch. This is to ensure that when one side is active the other is inhibited.

It will be necessary to prevent any scenario in which both the 'on' and 'off' sides of the switch are active simultaneously. This can be accomplished by further delaying the connections denoted in figure 4-4 as the 'on' and 'off' side excitatory links. Consider the case where the 'off' side of the switch is active and i emits a spike. We require that the 'off' side be inactive by the time 'on' is made active by the spike from i. Therefore we add a delay to the synaptic connection from i to 'on', while instantly relaying the multiple inhibitory spikes from i to 'off'. The result is that the internal state of the 'off' oscillator will be lowered by an amount that will prevent it from its next firing so that when the excitatory pulse arrives at 'on', the 'off' side will be quiet. The delay is such that the single pulse arrives after the influence of the mutual inhibitory links has subsided.

4.8.3 Elements of a synchroniser: The accumulator neuron

Once the presence of an input destined for the oscillator has been detected by the switch, it will be necessary for the synchroniser not only to 'remember' that there is an outstanding input which needs to be injected into the target oscillator, but also be able to relay a pulse into the oscillator at the appropriate instant.

The accumulator neuron received input from the switch and also from the oscillator A, as shown in figure 4-4. Suppose that the internal state function of the accumulator is $N(s)$. We may write $N(s) = O(s) + A(s)$, where $O(s)$ is the response to the oscillator and $A(s)$ is the response to the inputs from the switch. Since we suppose that the oscillator has been nearly periodic for a while, it follows that $O(s)$ is nearly periodic with the same period. We suppose that the link or links between the oscillator and the accumulator have been set so that $O(s)$ is not only periodic with the same period as the oscillator, but also has just maximal point in between each firing of the oscillator. Let θ be the threshold of the accumulator. Pick the weights so that the maximum value of $O(s)$ is $\theta/2+\delta$, where δ is some small value, well below $\theta/2$. $O(s)$ might have some other local maxima which are smaller than this but these must all be below $\theta/2$. The time interval in which $O(s) > \theta$ can be made as small as we like by choosing δ sufficiently small. When the switch is off, the accumulator will not fire, since $O(s)$ will not cross the threshold.

When the switch is on, we arrange so that $A(s)$ rises to a plateau of approximately $\theta/2 + \delta$ and oscillates around this with amplitude below δ.

This is accomplished by having n time delayed synapses which connect the 'on' side of the switch to the accumulator neuron. The time delays are $j\tau/n$, for $j = 1 \ldots n$, where τ is the period of the 'on' side of the switch.

The weights of these connections are all equal to some value w, which is determined so that the resulting cascade of pre-synaptic pulses has the effect of charging the accumulator neuron so that $A(s)$ reaches the desired plateau. The

more connections there are, the more closely will the course of $A(s)$ approximate to the ideal of perfect flatness.

The accumulator will only spike if it has received a sequence of inputs from the 'on' side of the switch and it subsequently receives a input from the oscillator. At this point the accumulator neuron 'injects' the spike into the oscillator, and sends inhibitory spikes to the 'on' oscillator followed by an excitatory spike to the 'off' oscillator. The charging of the accumulator from the switch can be seen in figure 4-5

4.8.4 Results for the synchroniser in CSIM

Figure 4-6 shows the behaviour of the synchroniser as a whole, while figure 4-7 shows the injection of the three spikes as shown before in figure 4-3, but this time utilising the synchroniser. It can be seen that with the synchroniser the spike that was ignored previously is now injected successfully.

Suppose that the synchroniser has been turned on, and that a spike from the oscillator occurs at time t, and a subsequent spike from the accumulator occurs at time $t + \Delta$. The value of Δ will depend on the phase of the oscillator.

We have that $A(t + \Delta) + O(t + \Delta) = \theta$. Let the maximum value of $O(s)$ be $\theta/2 + \delta$. So $A(t + \Delta) \geq \theta/2 - \delta$. On the other hand, $A(s)$ is bounded from above by $\theta/2 + \delta$. We have:

$$\theta/2 - \delta \leq A(t + \Delta) \leq \theta/2 + \delta$$

Let $t*$ be the maximum point of $O(s)$ which is nearest to $t + \Delta$.

The quantity $\mid t + \Delta - t* \mid$ is bounded by the time it takes the value $O(s)$ to go from $\theta/2 - \delta$ to its maximum value $\theta/2 + \delta$ and then back down to $\theta/2 - \delta$; and this tends to zero with δ since $O(s)$ is continuous. Therefore the time between $t + \Delta$ and the next subsequent spike of the oscillator is approximately constant.

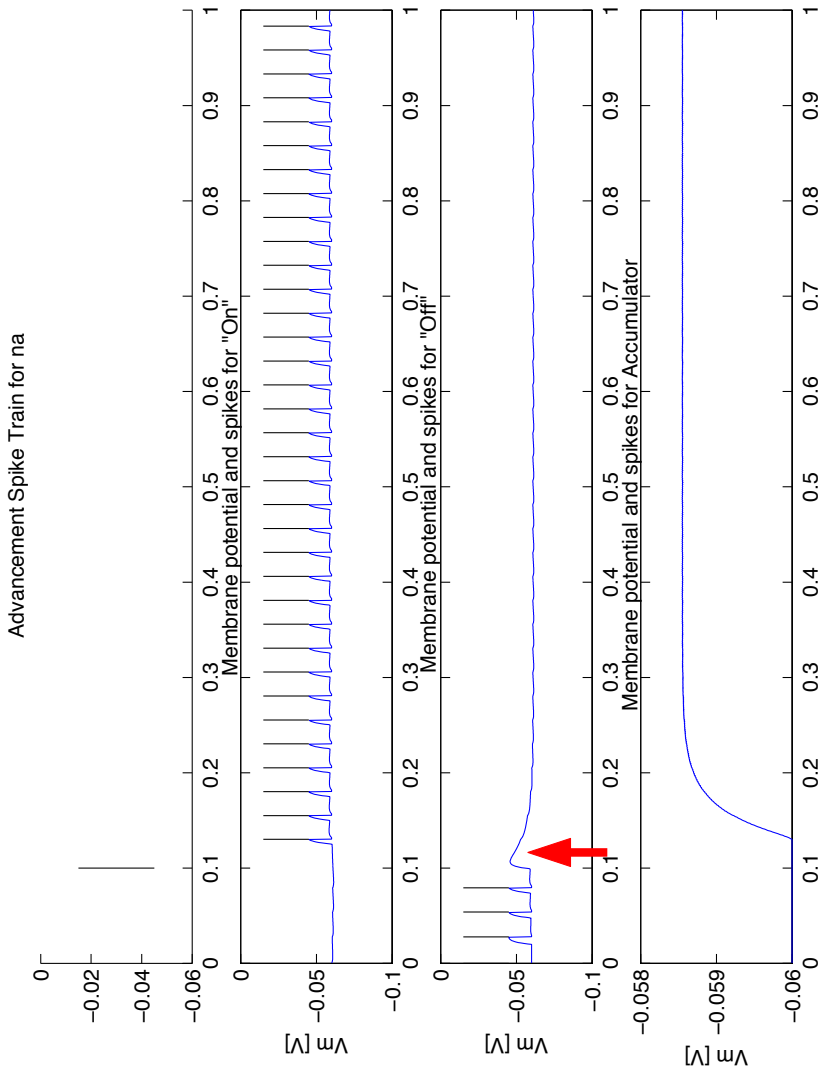


Figure 4-5: The Charging of the Accumulator. The third panel down shows the 'off' side of the switch set for firing. The top panel shows the advancement spike destined for an oscillating neuron, n_a. This spike is first sent the 'on' side of the switch. It can be seen that the advancement spike causes the 'off' side to cease firing and then the 'on' oscillator starts to fire. The switch is now on and the many outputs from the 'on' side of the switch begin to charge the internal state of the accumulator neuron, which can be seen in the bottom panel.

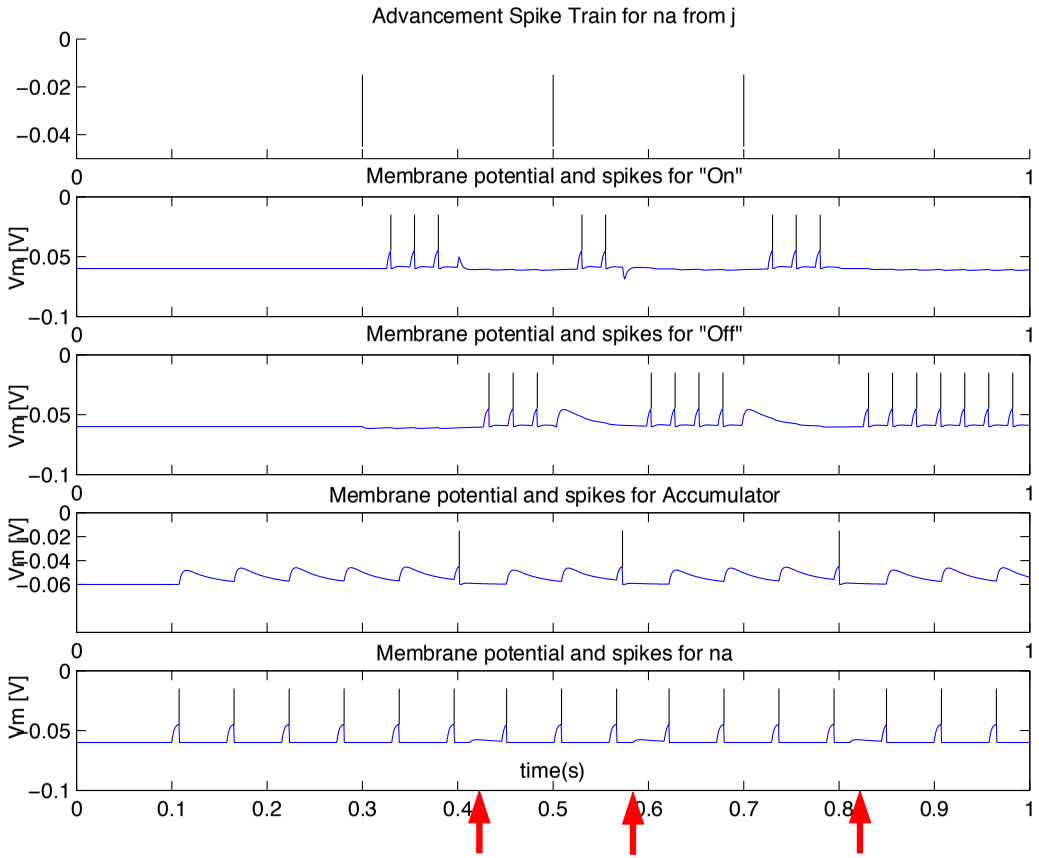


Figure 4-6: This figure demonstrates the behaviour of the various elements of the synchroniser while in use. The top panel shows three input spikes that are to be injected, via the synchroniser, into the oscillating neuron, n_a, whose output is shown in the bottom panel. The spikes can be seen to operate the 'on' and 'off' elements of the switch as described in figure 4-5. The accumulator, whose output is shown in the third panel down, then fires and injects each input spike into the oscillating target neuron at the same point in the phase of this oscillating neuron. The red arrows point to very slight variation in the internal state of the oscillator (blue line). These three slight 'bumps' in the internal state indications that the phase of the oscillator has been advanced each time.

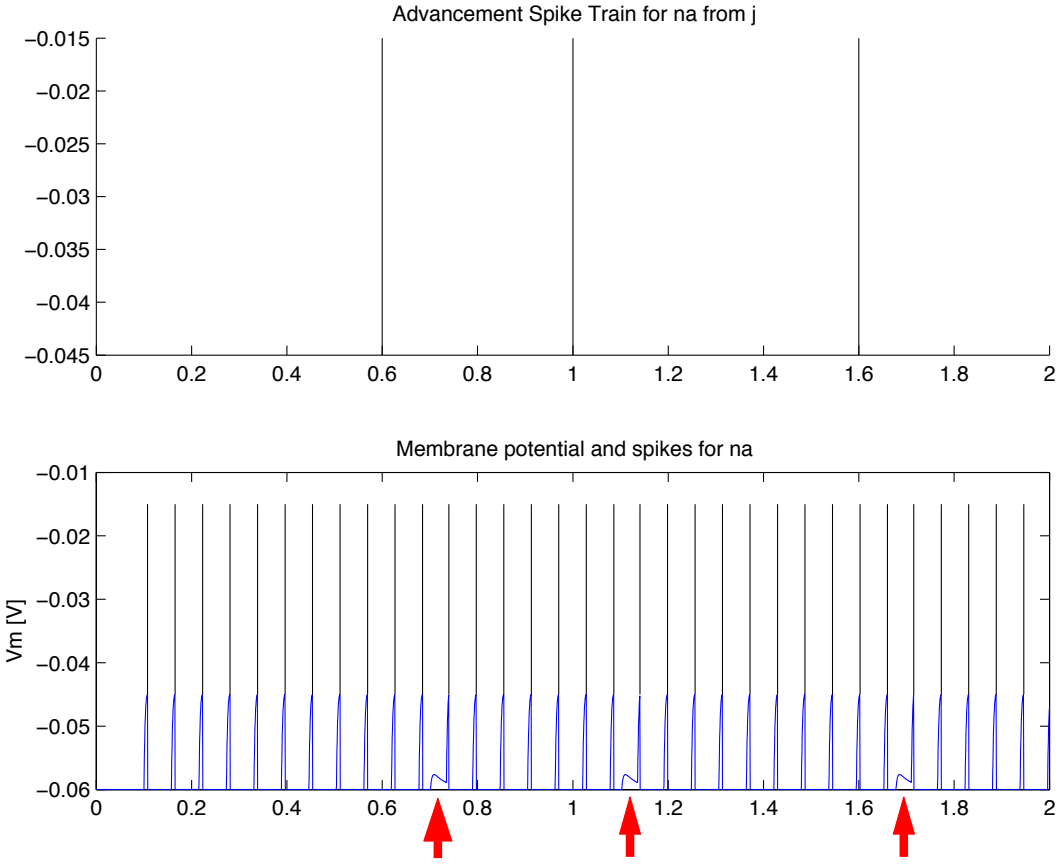


Figure 4-7: Using the same demonstration previously seen in figure 4-3, it can be seen here that, with the synchroniser the third spike now has exactly the same effect on the oscillator as the first two spikes. Using the synchroniser, input spikes are now injected into the oscillator neuron at the same point in its phase every time. Therefore, they all have the same effect on the oscillator.

When we advance the phase of the oscillator, the phase of $O(s)$ will follow, keeping approximately constant the distances between maximum points of $O(s)$ and the time of the next subsequent spike of the oscillator.

4.9 The Coincidence Detector

The function of a coincidence detector is to indicate if two or more spikes, from two or more separate inputs, are 'simultaneous' events. In any real system it is expected that simultanaeity can not be observed with perfect accuracy. Therefore, we define two spikes as being effectively simultaneous if they both arrive within a time interval of size ϵ, where ϵ is small.

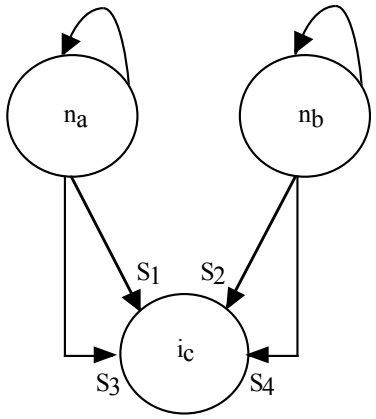


Figure 4-8: The structure of the coincidence detector.

It is expected that the coincidence detector will be highly susceptible to variations in the synaptic weights of each input. It will be necessary to choose the weights with particular care, as they will determine how selective the detector is at classifying two inputs as effectively simultaneous.

Consider a coincidence detector which is a single spiking neuron i_c that has two excitatory synaptic inputs s_1 and s_2 and two (slightly) time delayed inhibitory synapses s_3 and s_4, as shown in figure 4-8. Synapses s_3 and s_4 have the effect of sharpening the edge of the input spikes allowing the coincidence detector to be

more selective.

If the threshold of i_c is given by θ, and the synaptic weights of the two inputs are given by w_1 and w_2, where $w_1 = w_2 = w$ we set w so that

$$\theta = (2 \times w) - x$$

As x tends towards 0, ϵ will also tend towards 0. By adjusting the parameters, we can make ϵ as small as we desire.

4.9.1 Results for the Coincidence Detector in CSIM

In order to perform accurate operations with phase differences it is necessary that the integrity of a number stored in a pair of oscillators can be guaranteed.

Figure 4-9 shows the increment oscillator being perturbed 10 times by a phase advancement spike. This is effectively storing the number 10 in the pair. 10 spikes are then injected into the decrement neuron until both oscillators are in phase once again. The number of spikes required for this is 10. Therefore, integrity has been demonstrated in this case.

4.10 IF, WHILE, SEQ, PAR, ALT constructions

Given structures such as the oscillator, synchroniser and coincidence detector it is possible, using combinations of these structures, to implement the basic constructions of the programming language described above.

4.10.1 The IF constructor

Consider some group of spiking neurons which perform some process P. We are not concerned with the function or the mechanics of the process, only that it is

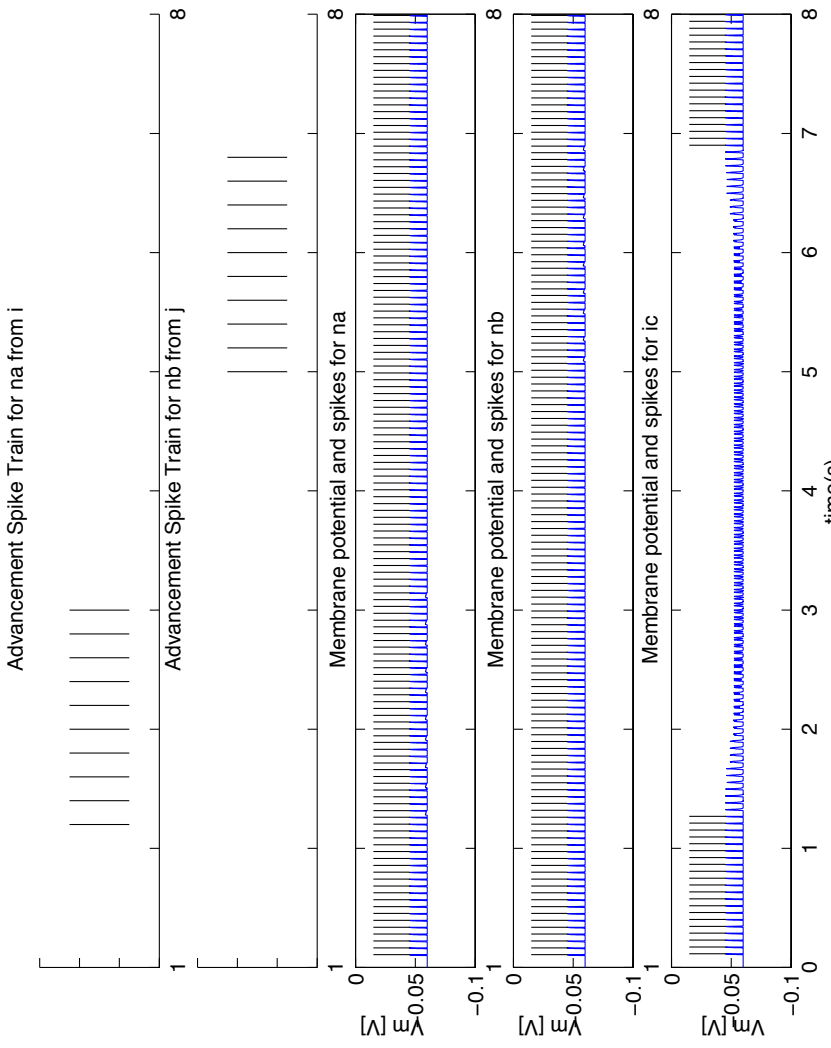


Figure 4-9: This integrity test demonstrates the coincidence detector neuron (bottom panel). The increment oscillator, neuron n_a, is perturbed by a series of ten spikes from spiking input neuron i (Top panel). The coincidence detector ceases firing as the increment and decrement oscillators are no longer synchronised. However, perturbing the decrement oscillator, neuron n_b, also by ten spikes from spiking input neuron j. The coincidence detector once again beings firing — the increment and decrement oscillators, n_a and n_b respectively, are once again synchronised.

implemented using spiking neurons and synapses, and that it can be triggered by a spike arriving from an input neuron i and that upon completion it activates some termination neuron which, effectively passes control to the next constructor.

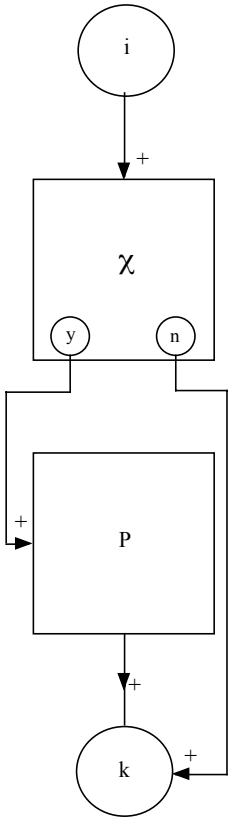


Figure 4-10: Example of the IF Constructor

χ is a structure of spiking neurons which represents the condition of the IF statement which, must be satisfied if P is to be activated. χ has an internal switch, not shown in the diagram, which is normally off. When this switch is off, neither output of χ can fire. Upon firing, the neuron i sends a pulse to the input neuron of χ, which then turns on χ by resetting its switch. Whether or not the condition is satisfied will determine which output neuron of χ fires. If χ is satisfied neuron y is activated, if χ is not satisfied then neuron n is activated. In either case, when χ finishes, it is switched off. Neuron y will send a spike to the initiating neuron of the process P that, upon completion will activate the

termination neuron of the constructor k. Neuron n will immediately activate k, as the condition for P to be activated has not been met.

A precondition for this to work correctly is that χ is turned off before the initiating neuron fires.

4.10.2 The WHILE constructor

This is similar in structure to the IF statement. The WHILE constructor must first check if a condition χ is met and then, if it is, allow a process P to be activated. The cycle continues until the condition represented by χ is no longer met. If at any point χ is not met when tested, then the WHILE constructor passes control to the next process. A possible implementation is shown in figure 4-11.

Again χ is the condition of the WHILE statement, P is the process to be run while this condition is met.

Neuron i is an activation neuron. A spike emitted by i will activate χ. Upon completion P emits a spike to the input neuron of χ. If the condition χ is being met then neuron y is activated, otherwise neuron n is activated.

Neuron y is connected to the activation neuron of the process P. If neuron n is triggered it will activate the constructor termination neuron k, passing control to the next constructor.

4.10.3 The SEQ constructor

The function of the SEQ constructor is to allow a series of processes to activate sequentially, with the next process in the list only being activated once the current process has completed.

Neuron i is the activation neuron of the constructor. $P_1, P_2, P_3 ... P_n$ is the sequence of processes to be activated. Assume that each of these has been implemented

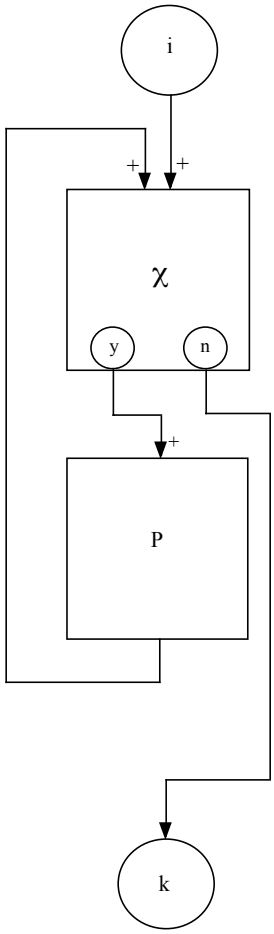


Figure 4-11: Example of the WHILE Constructor

by disjoint subsystems. After each process is completed only then will it send an activation spike to the next process in the sequence. As figure 4-12 shows, i sends an activation spike to P_1, which in turn sends an activation spike to P_2, which sends an activation spike to P_3, which upon its completion will send a spike that activates the next constructor.

4.10.4 The PAR constructor

The PAR constructor activates a list of processes concurrently. Assume, as before, that each of these has been implemented by subsystems which are disjoint.

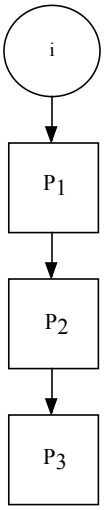


Figure 4-12: Example of the SEQ Constructor

Control only passes from a PAR constructor once *all* of the listed processes have completed.

Once again i is the activation neuron of the constructor. When i spikes, the spike gets transmitted to the input of each of the processes to be activated. As each process completes it will send a spike to its specific Input neuron n within an n input register, see figure 4-13. The weights of this switch are set, such that a constant charging from each of the n inputs is required to cause activation of the accumulator. The mechanisms for charging the accumulator are similar to those described in the previous section on the synchroniser, the only difference being that we now utilise multiple accumulator input oscillators. The activation of the accumulator causes a spike to be sent to the constructor termination neuron j, which passes control to the next constructor.

4.10.5 The ALT constructor

The ALT constructor (essentially nondeterministic choice) is given a set of processes implemented on disjoint subsystems, runs them concurrently, and terminates when one of the processes terminates. So the initiating neuron of the ALT

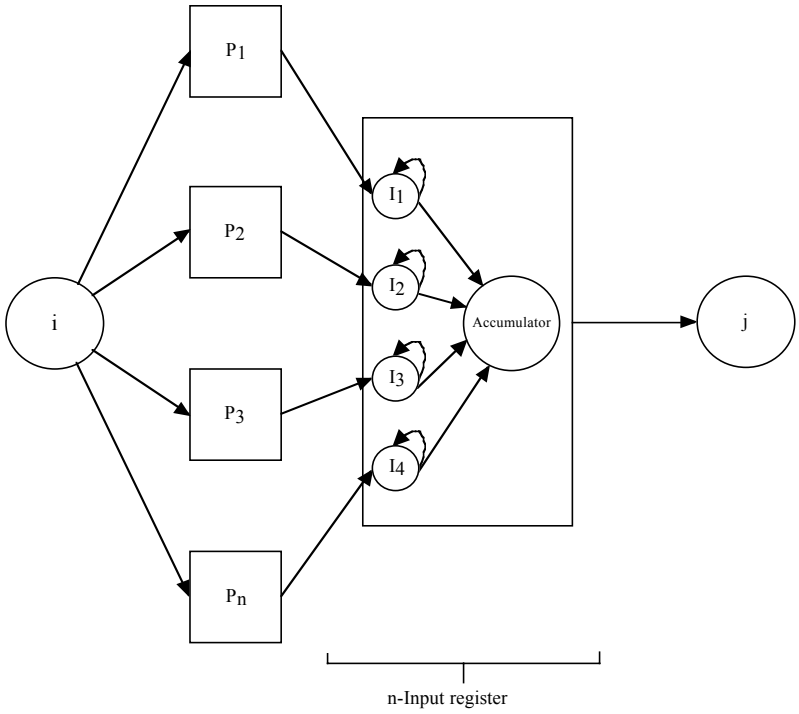


Figure 4-13: Example of the PAR Constructor

sends an initiating impulse to all of the initiating neurons of its constituent processes, and a spike from any of the terminating neurons of any of the constituent processes is sufficient to trigger a spike from the terminating neuron of the ALT. After the ALT terminates, it should turn itself off.

4.11 Examples: Addition, Subtraction and Multiplication

We can define addition, subtraction and multiplication over the integers in the language given earlier, using the recursive definitions starting from the successor function.

```
Seq{
  Z:=Y;
  while Z \= 0
  Par{
    Z:= Z-1;
    X:= X+1;
     };
    }
```

defines $X := X + Y$ if Y is non negative. Subtraction can be done similarly for Y non negative. Addition over the integers can be done as:

```
if Y > 0 then
Seq{
  Z:=Y;
  while Z \= 0
  Par{
    Z:= Z-1;
    X:= X+1;
     };
   }
else
Seq{
  Z:= Y;
  while Z \= 0
  Par{
    Z:= Z+1;
    X:= X-1;
     };
   }
```

To define $X := X * Y$, for Y non negative we can use

```
Seq{
  W:= Y;
  Wx:= X;
  X:= 0;
  while W \neq 0
  Par{
    X:= X+Wx;
    W:= W-1;
     };
   }
```

and then we can extend to the integers as was done for addition.

The inner Par constructions in the above statements could be changed to Seq constructions, without altering the function of the statements themselves.

4.12 Decidability and Complexity

If we had a perfect coincidence detector, i.e. sharp thresholds, and a perfect synchroniser, we could represent arithmetic for unbounded integers with a finite network. We could make such a network search for solutions to diophantine problems, which, as a class, are unsolvable. In the deterministic model therefore, long term behaviour would be expected to be undecidable. In fact even in very simple networks in the deterministic model, long term behaviour could be undecidable. This could be formalised using, for example, the Blum Shub Smale real number machine (Blum *et al*, 1997). If we add some noise to the links so that the fading memory concept (Maass *et al*, 2004b) applies, we may lose the undecidability, but we would still expect that in a practical sense the long term behaviour of these systems would not be feasible to compute.

4.13 Pointers and associations

It should be clear that it is also possible to represent pointers using networks of spiking neurons. Consider, for example, a pair of oscillators (A_p, B_p). We can say that this points to another pair (A_q, B_q) if B_p is in phase with A_q. In this way we can represent networks of associations. An attractive example of how such ideas can be used can be found in Wang (2000), where interactions of networks of oscillators are used to compute geometric and topological properties.

4.14 Conclusion and Discussion

The discussion above and the demonstrations we have done with CSIM suggest the possibility of a compiler which would take a statement in a simple parallel programming language such as the one given above and would write a script in for example the CSIM language which would construct a spiking neural network which would enact the computational meaning of the statement. Instead of training spiking neural nets to perform given tasks, we are constructing them, as is also done in Maass (1996). There are already examples of such compilers from Occam-like languages into sigmoid neural networks. See Neto *et al* (2003).

The implementation of addition, subtraction and multiplication which we give, for purposes of exposition, is very inefficient computationally. However, using the same basic ideas it should be clear that we can implement other, more efficient, algorithms for arithmetic. For example, we could implement a variable as an array or pairs of oscillators in such a way that when one of the pairs has been advanced as far as possible it returns to zero and the next pair is advanced by one step, as in the usual decimal notation. We could then build in an addition and multiplication table, and proceed, as we did, with effort, in school.

Chapter 5

Analysis of STDP Learning properties

5.1 Introduction

The work presented in this chapter is focussed on analysis, at a fundamental level, of a form of STDP learning within highly recurrently connected networks of spiking LIF neurons. In addition to STDP learning, the synaptic weights are also modified by a normalisation process. This normalisation allows for the norm of the weight vector to each network neuron to be maintained, and allows for the synaptic weights to assume any value between a minimum and maximum weight value. The reason for using such a normalising procedure is so that successive applications of STDP weight changes does not create a bimodal weight distribution in which the synaptic weights will eventually attain either the maximal or minimal possible value. The bimodal nature of such un-normalised STDP learning was shown in Song *et al* (2000). Using the normalisation means that the synaptic weights are able to assume a range of intermediate values between the minimum and maximum, making the learning meaningful and stable at these intermediate values in the case of long term application of STDP learning.

We show here that, the application of a form of STDP within a highly recurrent spiking neural net based upon the Liquid State Machine (LSM) leads to an approximate convergence of the synaptic weights. Convergence is a desirable property as it signifies a degree of stability within the network. This approximate convergence is not formally proved, but its presence is indicated experimentally. An *activity link* L is defined which describes the link between the spiking activity on a connection and the weight change of the associated synapse. This activity link L also enables perfect convergence to be defined as the point as which the input weight vector, W, of each neuron in a network are aligned with their respective activity link vectors. It is shown that under specific conditions Hebbian and Anti-Hebbian learning can be considered approximately equivalent processes. Also, it is shown that such a network habituates to a given stimulus and is capable of detecting subtle variations in the *structure* of the stimuli itself. It is also shown that it is possible to extend the firing duration of highly recurrent neural networks in a pre-determined manner. A method for storing precise spike trains within recurrent networks is also suggested.

There are forms of STDP learning other than the most basic Hebbian STDP learning. Two major disadvantages of basic Hebbian learning mean that it is unstable, see Bienenstock *et al* (1982): Firstly, the tendency is that weights increase or decrease exponentially with the end result being that synaptic weight values will lie only at the extremes of their operational range after sufficient exposure to learning; Secondly, basic Hebbian learning has no mechanism to allow the synaptic weights of unused connections to decrease in magnitude. In order to address these shortcomings of basic Hebbian learning, Bienenstock *et al* (1982) introduced the Bienenstock, Cooper & Munro (BCM) model of synaptic modification learning.

BCM theory incorporates modifications that make it a more favourable form of learning, over basic Hebbian learning. The BCM model includes a time constant that allows synaptic weights of unused connections to decay to some base weight value. Full details can be seen in Bienenstock *et al* (1982).

The first consideration of this chapter is to establish the details of the STDP

with normalisation ($STDP + N$) learning regime to be used on the synapses of the recurrently connected networks. The activity link vector L is then defined. Having defined the learning rule, it is then applied during a series of experiments. The initial result is that when a recurrently connected spiking neural network is stimulated by an external spiking source and the synaptic weights are modified using the $STDP + N$ learning that, the weights converge, as shown by a plot of the average synaptic weight change for the whole network w.r.t. the number of iterations. An explanation is then given, which describes how, under certain conditions, Anti-Hebbian learning is approximately equivalent to Hebbian learning. This approximate equivalence is then shown experimentally. Finally, it is shown experimentally that such a network with $STDP + N$ learning can become habituated to an input stimuli and that, even once this converged state of habituation is attained, the network is capable to respond to a change in the nature of the input stimuli by undergoing another sharp change in synaptic weights before once again converging and becoming Habituated to this new form of input stimuli. The reason for these basic experiments are to establish firstly that, the learning regime is stable, but more importantly, to investigate aspects of Hebbian and Anti-Hebbian learning, such as their approximate equivalence under certain conditions, that have not been considered in previous research.

5.2 $STDP + N$ learning

Spike Time Dependent Plasticity or STDP learning rules are based upon Hebb's postulate, see Hebb (1949). Suppose that a neuron n_{post} receives an input connection from neuron n_{pre}. Consider, that if the pre-synaptic neuron n_{pre} fires before the post-synaptic neuron n_{post}, then the synaptic weight on the link between them is strengthened. In this case n_{pre} can be thought of as contributing to the firing of n_{post}, so its influence is encouraged. If the firing sequence is reversed and n_{pre} fires *after* n_{post} then the connection is weakened.

The case when the synaptic weight is strengthened if n_{post} fires before n_{pre} and weakened if the firing is reversed is known as Anti-Hebbian STDP. The delay

between the firing time t_{pre} of n_{pre} and the firing time t_{post} of n_{post} is denoted as $t_{delay} = t_{pre} - t_{post}$ and is the determining factor in how large the synaptic weight change should be. A review of the importance of the precise timing of pre and post-synaptic spikes to synaptic modifications, as first outlined in Hebb (1949), can be found in Bi and Poo (2001).

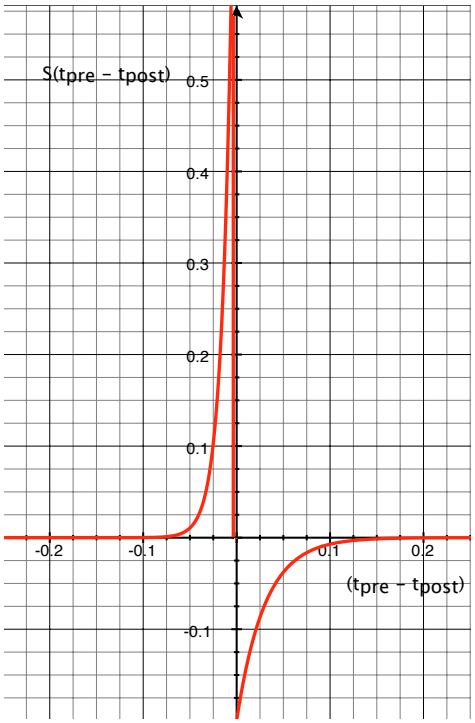
5.2.1 The $STDP + N$ Learning Window Function

In this chapter, an asymmetric Hebbian learning function is used, the form of which can be seen in figure 5-1. This is a typical learning function which increases those synaptic weights whose pre-synaptic neuron fires just before its post-synaptic neuron and weakens the synaptic weight if the post-synaptic neuron fires before the pre-synaptic neuron. The amount of the modification is determined according to the product of the learning window function $S(t_{delay})$ and some learning rate ϕ.

An Anti-Hebbian learning window function is one that operates in reverse to the Hebbian version. Therefore, synaptic weights are strengthened if the post-synaptic neuron fires before the pre-synaptic neuron, and weakened if the pre-synaptic neuron fires before the post-synaptic. In this implementation, for Anti-Hebbian STDP $-\phi$ is used for the learning rate instead of ϕ. It should be noted that there are different forms of STDP learning, see Izhikevich and Desai (2003), which shows that BCM learning can be derived from STDP learning under certain circumstances.

5.2.2 Weight Change and Normalisation

Consider a neuron Y within a recurrent neural network, similar to figure 2-1, where S is the learning window function, and which receives inputs from neurons $X_1, ..., X_k$ with respective synaptic weights $w_1 ... w_k$. W is the weight vector associated with the input of Y, $W = (w_1 ... w_k)$. The individual synaptic weights that comprise W are modified by two processes; a Hebbian weight update that



$$S(t_{pre} - t_{post}) = e^{((t_{pre} - t_{post})/0.011)} \quad \text{for } (t_{pre} - t_{post}) \leq -2.5ms$$
$$S(t_{pre} - t_{post}) = 0 \quad \text{for } -2.5ms < (t_{pre} - t_{post}) < 0ms$$
$$S(t_{pre} - t_{post}) = -0.2 * e^{(-(t_{pre} - t_{post})/0.03)} \quad \text{for } (t_{pre} - t_{post}) \geq 0ms$$

Figure 5-1: The form of the learning window function used to calculate the weight change due to STDP. The actual weight update value is given by the product of this learning function $S(t_{delay})$ and the learning rate ϕ. The time scale of the learning window function is such, that the peak occurs at a distance of $2.5ms$ from the post-synaptic firing time. This value was chosen in accordance with the finding in Gerstner and Kistler (2002), that the learning window function maximum $S(t_{delay})_{max}$ should be in the range $\delta/2 \leq S(t_{delay})_{max} \leq \delta$, where δ is the rise time of an EPSP, typically $5ms$. STDP is a biologically derived learning method, but there are many unanswered questions as to what actually happens when STDP is applied, especially in the context of recurrent spiking neural networks. An axonal time-delay is incorporated into a synaptic delay on the transmission of a spike to the receiving neruon. This synaptic time-delay was drawn from a gaussian distribution with a mean of $0.0015s$ for Excitatory-to Excitatory neuron connections, and a mean of $0.0008s$ for all other connections, and with a standard deviation of 10% of the mean.

is calculated for $w_1...w_k$, and a normalisation procedure which ensures that the input synapses to a neuron have a constant norm. Therefore, this a competitive Hebbian process, see Gerstner and Kistler (2002). The normalisation is applied to W only after all weights $w_1...w_k$ have had the Hebbian weight update procedure applied. Network activity is simulated in $0.0002s$ time-steps and weights are updated every $0.5s$ for the duration of the simulation. Therefore, using this update schedule, spikes are collected over a $0.5s$ interval and the input weights of each neuron are updated taking into account all pre and post-synaptic spike interactions within this period. The weights are *not* updated after each spike or neuron firing. This is a reasonable approach to take and can be justified if one considers that meaningful weight updates in biological neurons take place over a period of time and not necessarily with each firing of a neuron. These update procedures are performed on the weight vectors of each neuron of the network. Suppose R is the norm of the original unmodified weight vector W. For a neuron X_i firing at time t_i and a neuron Y firing at time τ, we get:

$$w_i := w_i + \phi S(t_i - \tau) \text{ for } i = 1, ..., k$$

$$W := R * W/||W||$$

The normalisation procedure is a necessary tool if the *polarisation* of synaptic weights that can occur with Hebbian learning is to be avoided. This polarisation effect is when the constituent weights of the weight vector assume either the highest or the lowest possible value after many repetitions of the same I/O stimulus/response at the pre and post-synaptic neurons. The normalisation procedure used here allows the synaptic weights to assume a range of intermediate values that fall between the minimum and maximum possible values. Additionally, the weight vector formed is also stable. The weight update for the j^{th} synaptic input weight to the i^{th} neuron, in which there are multiple spikes in both the pre and post-synaptic spike trains is implemented as follows:

$$w_j := w_j + \frac{1}{M} \sum_{m=1}^{M} \frac{\sum_{k=1}^{K} S(t_j - t_f^m)}{K}$$

This is performed for all input synapses to the i^{th} neuron. M is the number of spikes of the postsynaptic neuron, K is the number of spikes of the j^{th} input, t_j^k is the firing time of the k^{th} spike of the j^{th} input to i, t_f^m is the postsynaptic firing time of the mth of neuron i. The updated weight vector W_i of the i^{th} neuron, is now normalised as follows:

$$W_i := \frac{W_i}{norm(W_i)} \cdot norm_{req}$$

Where, $norm(W_i)$ is the ℓ^2-norm over the synaptic weights of W_i and $norm_{req}$ is the norm to which the weight vector is being maintained. The inclusion of a normalising effect means that the weight-vector update assumes a winner-take-all behaviour and the synaptic weights are then each able to assume any value within the full range of values available.

The learning rate allows the strength of the Hebbian learning to be *tuned*. A learning rate that is too high could cause weights to jump around far too much, and could cause instability instead of a stable learned weight vector. Alternatively, a learning rate that is too low could cause the weight update values to be so small as to have very little effect on the synaptic weight values, even over a long period of learning many iterations. A full list of learning, and network generation parameters can be found in section A.1.2.

5.2.3 Fixing the Sign of the synaptic weights

In CSIM (IGI Group, 2008), a LIF neuron can be one of two types — either excitatory or inhibitory. The type of the neuron affects several parameters. Inhibitory neurons have a faster response time, to reflect the observed nature of biological excitatory and inhibitory neurons, and can therefore have a somewhat dominating effect on network behaviour. The parameters for each type of neuron are based on data obtained from study of examples biological neurons (Thomson *et al*, 2002).

Finally, in addition to the parameters already mentioned, the most major difference between the two types is the sign ascribed to the synaptic weights of the synapses to which a given type of neuron is connected. The synapses receiving connections from *excitatory* neurons are positive, while synapses receiving connections from *inhibitory* neurons are assigned a negative weight.

In biological neural networks, the type of a neuron can be generally described as inhibitory or excitatory. The type of a neuron is a fixed attribute. Indeed, there are marked physiological differences between the two different types of neurons, (Peters and Jones, 1984). In addition to differences in connectivity, size and shape there are also differences in the type of neurotransmitter chemical employed at the synapse. Whether a particular connection is inhibitory or not also appears to be dependent not just on the type of the pre-synaptic neuron and the neurotransmitter chemical used, but also by the type and response of the post-synaptic receptor to the pre-synaptic neurotransmitter.

Evidence has also been found that suggests that the type of a synapse can change from excitatory to inhibitory (Fransén, 2005).

The implementation used for $STDP + N$ that has been outlined is one in which the sign of a synapse (excitatory or inhibitory) is preserved. If a synapse whose pre-synaptic neuron is positive i.e. excitatory, were to have a negative synaptic weight modification applied to it that is of sufficient magnitude to change the sign of the synapse from positive to negative then, the synaptic weight is set equal to zero.

This solution successfully addresses the problem of synaptic weight changing sign during $STDP+N$ learning, however it also creates the possibility of another problem arising. Consider, that the learning rate of the $STDP + N$ implementation has a direct effect on the absolute value of the synaptic weight modification calculated by the learning function. Now consider that a synapse with a positive synaptic weight that is to have a large enough negative change applied to it could have the synaptic weight set to zero. This means that if a learning rate that is too large is applied, it would then be the case that a large number of synaptic

weights would be set to zero. Therefore, in order for the learning function to work correctly with the set-to-zero method just outlined, the learning rate would need to be small enough to allow meaningful weight changes to be made without setting a large proportion of the synapses to zero.

Weight fixing was not used for the experiments of chapter 5. The reason for this was that allowing the weights to change sign means that the weight vectors are able to become better aligned with the activity link vectors than if weights are clipped to zero when a weight update may prompt a sign change.

5.2.4 Application of $STDP + N$ learning

In terms of application of $STDP + N$ for a given network, N, of recurrently connected neurons, the learning rule is applied sequentially to all neurons within N.

Consider the following scenario: A network of recurrently connected LIF neurons receives an input stimulus from a separate set of *input* neurons. The spiking input activity is injected into the network and the spiking activity of the neurons of the network is simulated for the time period $[0, Tsim]$, where $Tsim$ is the duration of the simulation. The input and output spike trains of each network neuron are recorded and each neuron of the network is then processed by the learning function seen in figure 5-1. The network neuron being dealt with at any instance is treated as the post-synaptic neuron.

For each spike contained in the spike train of the post-synaptic neuron, $STDP+N$ learning is done sequentially, between each post-synaptic spike and each spike on every *active* input channel to the post-synaptic neurons. An active input channel is a channel that has a pre-synaptic neuron whose output spike train is not empty. The neurons are processed in a sequential manner in this way until all have been dealt with.

5.3 Activity Link

In order to better describe the weight change brought about by the Hebbian update, and as a means for describing the link between the spiking activity on a synaptic connection and the weight change of the synapse associated with that connection, the activity link $L(a, b)$ is defined where a and b are spike trains. Suppose that a has spike times $t_1, ..., t_m$ and b has spike times $\tau_1, ..., \tau_n$, then we can define the following:

$$L(a, b) = \sum_{i:=1}^{m} \sum_{j=1}^{n} S(t_i - \tau_j)$$

Consider neuron Y from the previous section. Suppose that the spike trains of $X_1, ..., X_K$ are $x_1, ..., x_k$ and Y has a spike train y over an interval I. The update rules for the Hebbian learning followed by the normalisation update are:

$$w_i := w_i + \phi L(x_i, y) \text{ for } i = 1, ..., k$$

$$W := R * W/||W||$$

5.4 LSM generation parameters

All experiments are performed using CSIM, see IGI Group (2008), under MATLAB. CSIM allows for the simulation of many types of neuron and synapse models, and enables the creation of pools of recurrently connected neurons.

For the sake of completeness, the choice of the major parameters are given below.

Connection Probability — This is determined by the term $C \cdot e^{-D^2(a,b)/\lambda^2)}$: λ determines the average euclidean distance between the neurons as well as the average number of connections, the euclidean distance between two neurons a and b is given by $D(a, b)$, and the value of C is dependent on whether a and b are excitatory (E), or inhibitory (I). Therefore C is 0.3(EE), 0.2 (EI), 0.4 (IE),

0.1(II). This is taken from Maass *et al*, see Maass *et al* (2004a).

Post Synaptic Potential — A Post Synaptic Potential (PSP) can be considered to be the post-synaptic *effect* of a post-synaptic spike on the internal state of the target neuron. The form of a PSP is $e^{(-t/\tau)}$. For an Excitatory Post-Synaptic Potential, EPSP,$\tau = 3ms$ for excitatory and $\tau = 6ms$ for an Inhibitory Post-Synaptic Potential, IPSP. In this model, the rise time of the EPSP/IPSP is $5ms$.

5.5 Experiments

5.5.1 Experimental Setup

A recurrently connected network of 135 LIF neurons in a 5x3x9 arrangement is created with connectivity parameters $\lambda = 1.2$ and C was chosen according to the type of connection as stated previously, 0.3(EE), 0.2 (EI), 0.4 (IE), 0.1(II). A single spiking input neuron is connected to a random selection the other 134 total neurons that comprise the network. The connectivity parameters for connecting the input neuron to the network are set such that the input neuron connects to between 10% and 20% of the recurrent network neurons. For a full list of parameters used please see appendix A.

An input spike train is generated as follows. For each time-bin there is a probability of 0.5 of getting a burst of spikes at a frequency of $150Hz$, and a probability of 0.5 of getting no spikes at all. The time-bin duration is arbitrarily chosen to be $20ms$.

5.5.2 Synaptic Weight Convergence

10 Networks were generated using the above parameter values. Hebbian STDP was then applied to the internal weights of the network while the network was stimulated by the input neuron for a period of $300s$. It can be seen in figure

5-2 that the mean synaptic weight change over all networks undergoes a series of large initial changes, before eventually settling down to a stable state. This state is referred to as a state of approximate convergence and can be thought of as a state in which the synaptic weights of each neuron have become approximately aligned with the pre and post-synaptic firing activity.

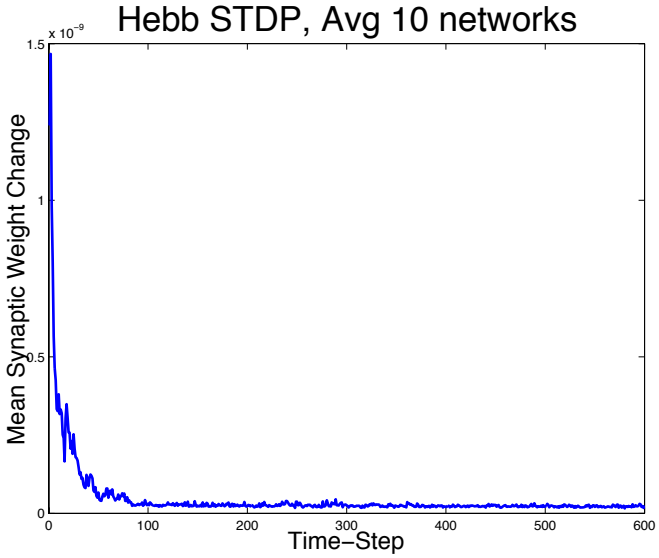


Figure 5-2: A graph of the absolute mean synaptic weight change due to Hebbian $STDP + N$ learning, averaged over 10 different networks. The networks are exposed to a stimulus of duration $300s$.

Consider the concept of the activity link defined in section 5.3. The input weight vector of each neuron in the recurrent network is bounded by the normalisation procedure described in section 5.2.2. As an analogy, the weight vector for each neuron in the recurrent network, can be said to be constrained to a multidimensional sphere of radius $||W||$, see figure 5-3. The 'orientation' of a weight vector is affected by changes to its constituent individual synaptic weights.

Recall neuron Y and its input spike trains $x_1, ..., x_k$. Figure 5-3 illustrates the weight vector of Y, $W = (w_1, w_2, ..., w_k)$ when it is modified under Hebbian and Anti-Hebbian learning. In the case of Hebbian learning in figure 5-3, W is updated as described in section 5.2.2. A small value of learning rate ϕ is assumed. It is important that ϕ is not too large as this would cause synaptic weight changes

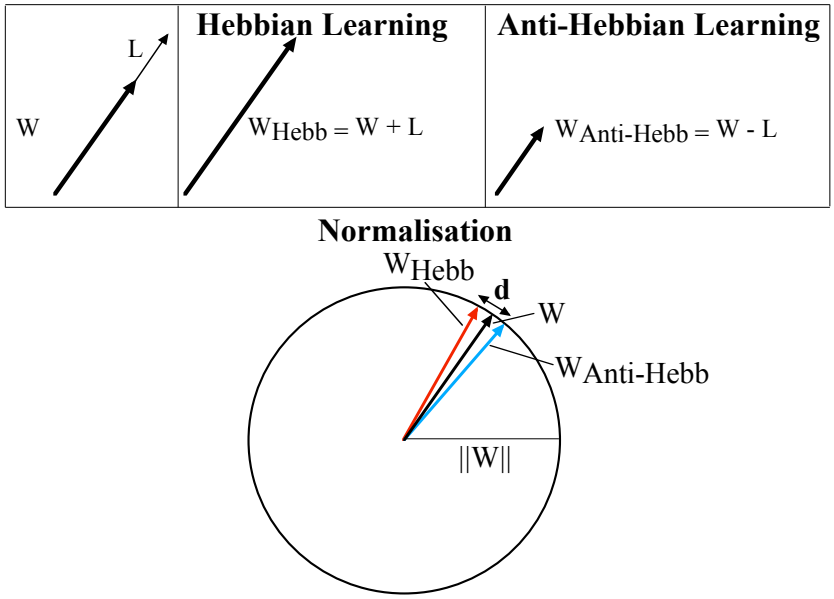


Figure 5-3: An illustration of how Hebbian and Anti-Hebbian learning are approximately equivalent, i.e. d is small, if W and L are approximately 'aligned' and if the learning rate ϕ is small.

to become erratic, and meaningful changes would be less likely to occur.

An approximate fixed point/stable state, of this Hebbian process is defined as being when the weight vectors of all neurons are approximately aligned with their activities. For neuron Y this means W is approximately aligned with the activity link vector $L = (L(x_1, y), L(x_2, y), ..., L(x_k, y))$. Given this definition of fixed points and of the activity link it can be seen that when ϕ is small and W and L are approximately in the same 'direction', the modifications to W caused by Hebbian learning followed by normalisation are approximately equivalent to modifications by Anti-Hebbian learning — see figure 5-3.

The variable d in figure 5-3 describes the 'difference' between W after Hebbian and W after Anti-Hebbian learning. d will tend to be small when W and L are approximately in the same direction and if ϕ is sufficiently small, in which case Anti-Hebbian learning will have the same approximate stable states as Hebbian learning for a given recurrent network. We believe that for any single network there may exist many of these stable states, each of which could be maintained

by Hebbian or indeed, Anti-Hebbian learning.

5.5.3 Hebbian and Anti-Hebbian Approximate Equivalence

The aim of the work in this section is to experimentally test the statement that, under certain conditions and given enough Hebbian learning iterations, subsequent Hebbian and Anti-Hebbian learning can be considered approximately equivalent. Consider the a similar experiment described above but in which, a network undergoes Hebbian learning with much smaller learning rate for an extended number of iterations — see section A for details. The reason for this is to ensure that the synaptic changes are small enough that a sufficiently accurate alignment of W and L can occur. The left-hand panel of figure 5-4 shows the actual weight values of the weight vector of one of the network neurons after 1000 iterations of Hebbian learning. This network is then exposed to Anti-Hebbian learning, also for 1000 iterations, and figure 5-4 also shows the same weight vector after the additional 1000 Anti-Hebbian iterations. It is seen that there are obvious visual differences between the values of the weight vector after the Hebbian and then the Anti-Hebbian learning. This alternate application of Hebbian followed by Anti-Hebbian learning was repeated over 10 networks.

The same experiment was performed again, but for 5000 iterations of Hebbian learning, followed by 1000 iterations of Anti-Hebbian learning, also for 10 networks. It can be seen in figure 5-5 that the differences between the weight vector of a randomly chosen neuron after Hebbian learning and after subsequent exposure to the Anti-Hebbian learning, is negligible. The Anti-Hebbian learning does not appear to have significantly affected the individual values of the weight vector.

The left hand plot of figure 5-6 shows the average difference caused in the synaptic weights of the input weight vector of each network neuron, due to applying 1000 iterations of Anti-Hebbian learning to networks that have previously been exposed

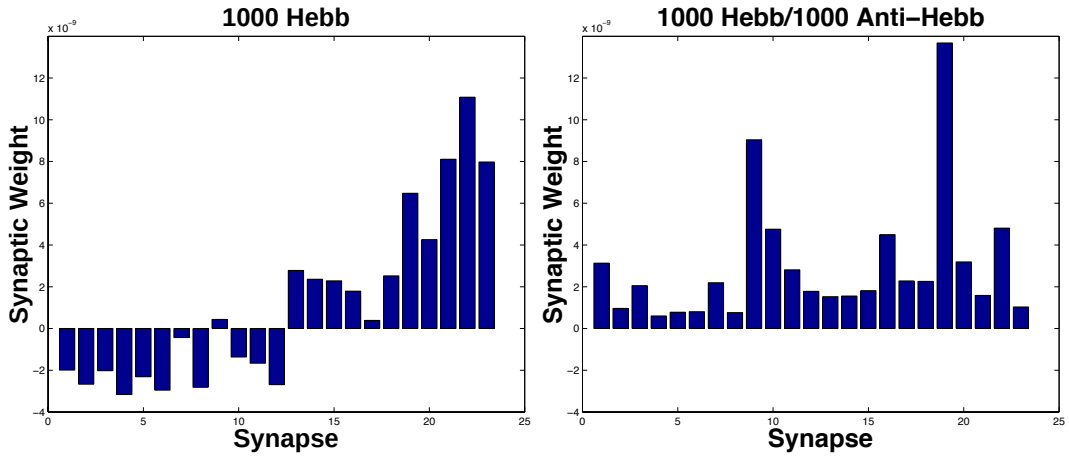


Figure 5-4: Weight vector values for a single network neuron. The left-hand panel shows the weight vector after 1000 iterations of Hebbian learning. The weight vector has mean, $m = 1.15e^{-09}$, and standard deviation, $s = 4.10e^{-09}$. The right-hand panel shows the weight vector after the network has undergone subsequent learning of 1000 iterations under an Anti-Hebbian learning regime, this weight vector has $m = 2.95e^{-09}$, $s = 3.01e^{-09}$. It can be seen that the weight vector has undergone some noticeable changes, with some weights changing their sign and their strength by relatively large amounts. The absolute values for the change in the mean and standard deviation are $|\Delta m| = 1.80e^{-09}$ and $|\Delta s| = 1.09e^{-09}$ respectively.

to 1000 iterations of Anti-Hebbian learning. The average change in the weight vector is denoted as δW^i_{Avg} for a neuron i, averaged over 10 networks. If $W^i_{Hebb} = (w^H_1, w^H_2, w^H_3, ..., w^H_k)$ is the weight vector of a network neuron i after undergoing Hebbian learning and, $W^i_{Anti} = (w^A_1, w^A_2, w^A_3, ..., w^A_k)$ is the weight vector of neuron i after it has undergone subsequent, Anti-Hebbian learning then, for this single neuron i, $\delta W^i_{Avg} = \frac{\sum^k_{j:=1} w^H_j - w^A_j}{k}$.

The right-hand plot of figure 5-6 shows δW^i_{Avg} for each neuron i, averaged over 10 networks, but for 5000 iterations of Hebbian learning prior to the 1000 iterations of Anti-Hebbian learning. It can be seen that this average change is markedly lower that the typical changes seen in the case in which the network was only exposed to 1000 iterations of Hebbian learning before being exposed to the 1000 iterations of Anti-Hebbian learning.

These results would appear to indicate that, if a network is trained for a period

of time that allows it to reach an approximately stable/converged state, then the changes to the weight vector that are effected by either Hebbian or Anti-Hebbian learning are approximately equivalent. Convergence was initially determined by a visual inspection of the plots of average synaptic weight change w.r.t. iterations of the learning procedure. However, convergence has also been more precisely defined as being when the weight vectors of each neuron in a network are *aligned* with their respective activity link vectors. This alignment is defined in section 5.6 of this chapter and provides a quantitative measure of convergence. It would also appear that this result suffers from little deviation or divergence as learning time (number of iterations), increases. It can be seen in figure 5-5, that 1000 iterations of Anti-Hebbian learning does not alter the weight vectors significantly from the learned stable state that resulted from 5000 iterations of Hebbian learning. However, it is possible that yet further application of Anti-Hebbian learning could yield a divergent behaviour of the weight vectors of the network neurons that could adversely affect the alignment of the weight and activity link vectors for each of the network neurons.

If the network is not allowed to reach an approximate stable state from the initial Hebbian learning phase, then the Anti-Hebbian learning can effect significant changes upon the weight vectors of the network. These results would seem to indicate that it may be the case that in order to maximise the ability of a network to learn, learning should take place early in the development of synaptic weights in response to continued repetition of a stimulus. It should be noted that the implementation of $STDP+N$ modification here, does not contain a term to allow a synaptic weight to decay with time, in the absence of any stimulus, as is the case in the BCM model, (Bienenstock *et al*, 1982).

5.6 Measuring Alignment

The experiments shown in the previous section demonstrate that exposing a network to many iterations of Hebbian learning appears to minimise the effect of Anti-Hebbian learning. However, they do not relate the alignment of W and

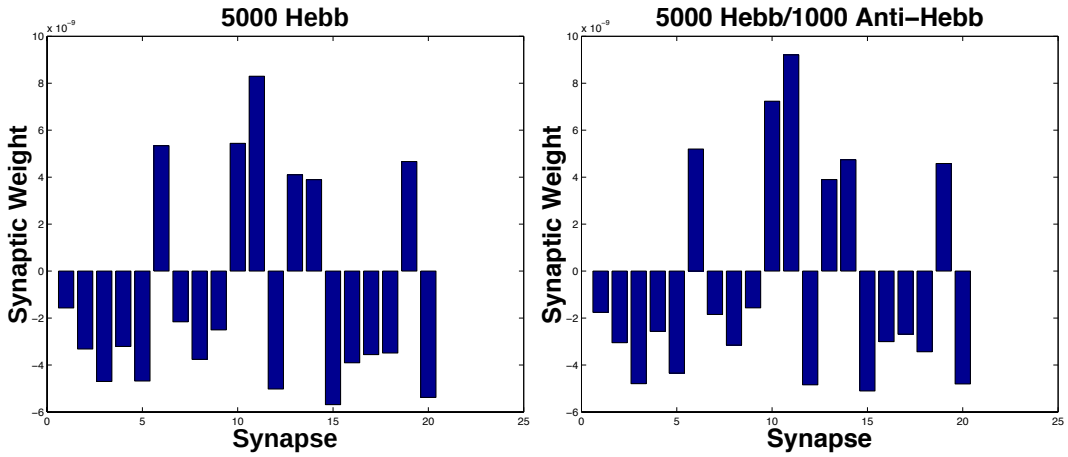


Figure 5-5: Actual weight vector values for a single network neuron. The left-hand panel shows the weight vector after 5000 iterations of Hebbian learning. The weight vector has mean, $m = -1.05e^{-09}$, and standard deviation, $s = 4.46e^{-09}$. The right-hand panel shows the weight vector after the network has undergone subsequent learning of 1000 iterations under an Anti-Hebbian learning regime. This weight vector has mean, $m = -6.04e^{-10}$, and standard deviation, $s = 4.55e^{-09}$.It can be seen that the weight vector has remained essentially unchanged. The absolute values for the change in the mean and standard deviation are $|\Delta m| = 4.56e^{-10}$ and $|\Delta s| = 9.00e^{-11}$ respectively.

L for the network neurons — as illustrated in figure 5-3 — to the approximate equivalence of Hebbian and Anti-Hebbian learning. The alignment of W and L shall be referred to as A, and the equivalence of Hebbian and Anti-Hebbian learning, is investigated as follows.

Consider, the same type of 135 neuron networks used in the previous experiments in this chapter, stimulated with the same type of input spike train. The networks are simulated for 10 runs each of 1000, 2000 and 5000 iterations of Hebbian learning. For each run, after Hebbian learning the activity link vector and the input weight vector are calculated for each neuron in the network. Let W_i and L_i be the weight vector and activity link vector, respectively, of the i^{th} neuron of the network, and given as:

$$W_i = (w_1^i, ..., w_k^i)$$

$$L_i = (L(x_1^i, y^i), L(x_2^i, y^i), ..., L(x_k^i, y^i))$$

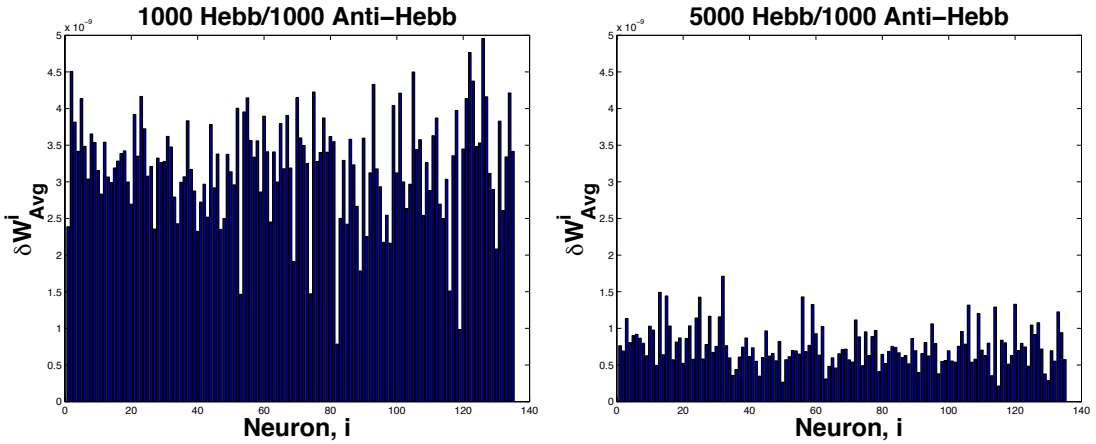


Figure 5-6: δW^i_{Avg} for Hebbian learning followed by Anti-Hebbian learning, in the case of 1000 and 5000 iterations of Hebbian learning each followed by 1000 iterations of Anti-Hebbian learning. Avg over 10 networks. This figure serves to illustrate the difference that 1000 iterations of Anti-Hebbian learning has on the weight vectors of networks that have already undergone Hebbian learning, in the cases of 1000 iterations, and of 5000 iterations of Hebbian learning. For the left-hand plot the mean and standard deviation are $m = 3.22e^{-09}$ and $s = 7.13e^{-10}$ respectively, while for the right-hand plot $m = 7.56e^{-10}$ and $s = 2.73e^{-10}$.

Where the i^{th} neuron receives k inputs, y^i is the output spike train of the i^{th} neuron, and $(x^i_1, x^i_2, ..., x^i_k)$ are the input spike trains for each of the k inputs of the i^{th} neuron. The alignment of W and L, after Hebbian learning, is then calculated for each network neuron. After Hebbian learning the weight vector of the i^{th} neuron is given by W^i_{Hebb} and and the activity link vector is given by L^i_{Hebb}, while the alignment of these two vectors is A^i_{Hebb} and is given by:

$$\frac{1}{A^i_{Hebb}} = norm(\frac{W^i_{Hebb}}{norm(W^i_{Hebb})} - \frac{L^i_{Hebb}}{norm(L^i_{Hebb})})$$

The network is then subjected to Anti-Hebbian learning for 1000 iterations, after which the alignment of the weight vector of the i^{th} neuron, W^i_{Anti}, with L^i_{Hebb} for all i network neurons is calculated and is given by A^i_{Anti} for the i^{th} neuron.

$$\frac{1}{A^i_{Anti}} = norm(\frac{W^i_{Anti}}{norm(W^i_{Anti})} - \frac{L^i_{Hebb}}{norm(L^i_{Hebb})})$$

Please note that the link vector used is from the Hebbian process and not the Anti-Hebbian link vector. This is done so that the alignment of W^i_{Anti} and L^i_{Hebb} can be compared to the alignment of W^i_{Hebb} and L^i_{Hebb}. The use of L^i_{Hebb} in both expressions provides a common factor that allows for comparison of any change in weight vectors.

Starting with a network that has undergone Hebbian learning and in which, the alignment of the weight vector with the activity vector for the i^{th} neuron is given by A^i_{Hebb}. The absolute change in alignment caused by the Anti-Hebbian process, A^i_Δ, w.r.t. A^i_{Hebb}, for the i^{th} network neuron is given by:

$$A^i_\Delta = \frac{|A^i_{Anti} - A^i_{Hebb}|}{A^i_{Hebb}}$$

Where A^i_{Anti} is the alignment of the weight vector and the activity link vector of the i^{th} neuron after the Anti-Hebbian learning. A^i_Δ is calculated for each network neuron and then averaged over all network neurons to give A^{Avg}_Δ. The average alignment for all neurons in a network after Hebbian learning is denoted by A^{Avg}_{Hebb}. Figure 5-7 shows a scattergram plot of A^{Avg}_Δ w.r.t. A^{Avg}_{Hebb} for 10 runs each of 1000, 2000 and 5000 iterations of Hebbian learning, each followed by 1000 iterations of Anti-Hebbian learning. It can be seen that the clear general trend appears to be that, as the alignment before Anti-Hebbian learning (A^{Avg}_{Hebb}) increases, the effect of the Anti-Hebbian learning on the weight vectors of the network neurons (A^{Avg}_Δ) decreases.

5.7 Hebbian Learning to Demonstrate Habituation

Using the same experimental setup as in the previous experiment, the stimulus spike train is now generated such that its frequency varies as a function of time. More specifically, the number of spikes, f, that occur in any $20ms$ time bin of the input spike train, is given by $f = C \cdot sin(t/t_c) + C$. Where t_c is the time

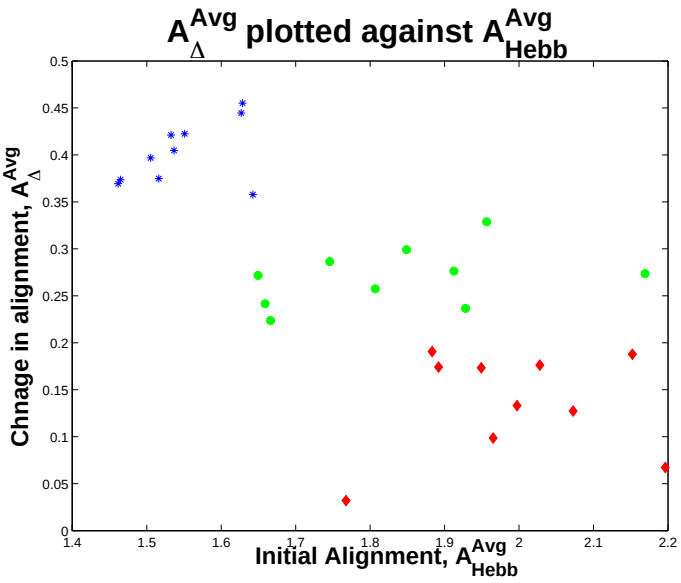


Figure 5-7: Scatter plot showing that the higher the average alignment, A_{Hebb}^{Avg}, of W and L before Anti-Hebbian learning, then the less effect Anti-Hebbian learning has on the change in alignment, A_{Δ}^{Avg}. The three different clusters represent the results of the exposure to 1000, 2000 and 5000 iterations of Hebbian learning and then 1000 iterations of Anti-Hebbian learning, over 10 randomly generated networks for each scenario of Hebbian learning. Blue represents 1000 iterations of Hebbian learning, green represents 2000 iterations of Hebbian learning and red represents 5000 iterations of Hebbian learning.

constant of the function, and C is a scalar which determines the range over which f varies. A spike train, S_1, is created according to $f_1 = 4 \cdot sin(t/2) + 4$. This stimulus is then used to drive the recurrent network to which the Hebbian learning window function is applied. The mean change in synaptic weights over the whole network is calculated and recorded at each $0.5s$ time step as before. The network is subjected to this stimulus for a duration of 200 seconds.

A second input spike train, S_2, is now created using a similar function to the first but with a time constant $t_c = 10$. So, $f_2 = 4 \cdot sin(t/10) + 4$. The same recurrent network that was subjected to S_1 for 200 seconds is now subjected to S_2 also for 200 seconds and again the mean change in synaptic weight over the whole network is recorded at each time step. It can be seen in the top left panel of figure 5-9 that the mean weight change undergoes a large initial change and

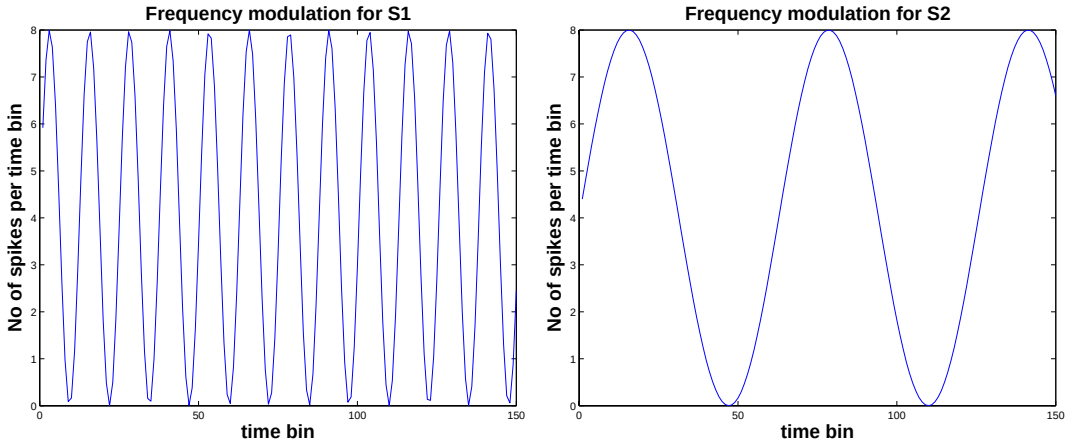


Figure 5-8: Plots of the different frequency modulation functions f_1 and f_2 used to create the input spike trains S_1 and S_2. The left panel shows a plot of $f_1 = 4 \cdot sin(t/2) + 4$. The right panel shows a plot of $f_2 = 4 \cdot sin(t/10) + 4$.

then settles down. At the onset of the change in the frequency modulation of the stimulus from S_1 to S_2 (time-step 400) it can be seen that the mean weight change once again undergoes a sharp increase and again settles down. If this same network is then re-presented with S_1, see top right panel of figure 5-9, the network does not respond with any kind of sharp change in synaptic weights. Similarly if the network is again presented with a stimulus drawn from S_2 (from time-step 400 in top right panel of figure 5-9), the sharp change in weights is also absent. Figure 5-8 shows a plot of f_1 and f_2.

This is an interesting occurrence because it would appear that upon being subjected to the stimulus with frequency modulation f_1 the network becomes habituated to this stimulus after prolonged exposure. When the frequency modulation is changed to f_2, the network is able to detect this subtle change and responds with an alteration of synaptic weights so that the weight vectors become aligned with the activity links once again. However, the weights are apparently altered in such a way that the learning of the frequency modulation of S_1 is not forgotten. This can be seen in the top right panel of figure 5-9 where S_1 doesn't provoke another series of sharp changes to the mean weight. This effect was observed by frequency modulation on the spike train stimulus of only a single spiking input

neuron.

Future work could extend this to investigate how many frequency modulations a network of given size can habituate to from a single input neuron. Additionally, it is thought that the introduction of multiple input neurons could increase number of 'Habituations' a given network could learn.

It can be seen in the lower panel in figure 5-9 that, as the difference between the time constants of the two input decreases, i.e. the two stimuli become more similar, the network is less able to respond in a distinct manner to each stimulus as expected. However, it is not yet known precisely how subtle a change in input stimuli is detectable by a given network. In order to determine this one could present many networks with a wide range of varying input types with a wide variety in similarity as measured by some metric. The response of the networks to switching between the patterns (of varying similarities) being presented to it could then be recorded.

5.8 Discussion

The first result of this chapter is the development and implementation of $STDP+N$ learning with normalisation. The properties of this form of learning are then investigated and can be summarised as follows:

The results shown in figure 5-2 clearly demonstrate that the $STDP + N$ Hebbian learning outlined in this section, when applied with appropriate learning parameters, facilitates the approximate convergence of the synaptic weights of the recurrent network. In such a case, the weight vector of each neuron becomes aligned with the activity on its own inputs. This means that the network has reached some stabilised state where the values of the synaptic weights have been determined by the spiking input activity into the network. Figure 5-2 shows mean weight change data averaged over 10 networks. In each case convergence is seen, even though the networks are generated in a random manner. This would seem to indicate that the convergence process is quite robust to variations in network

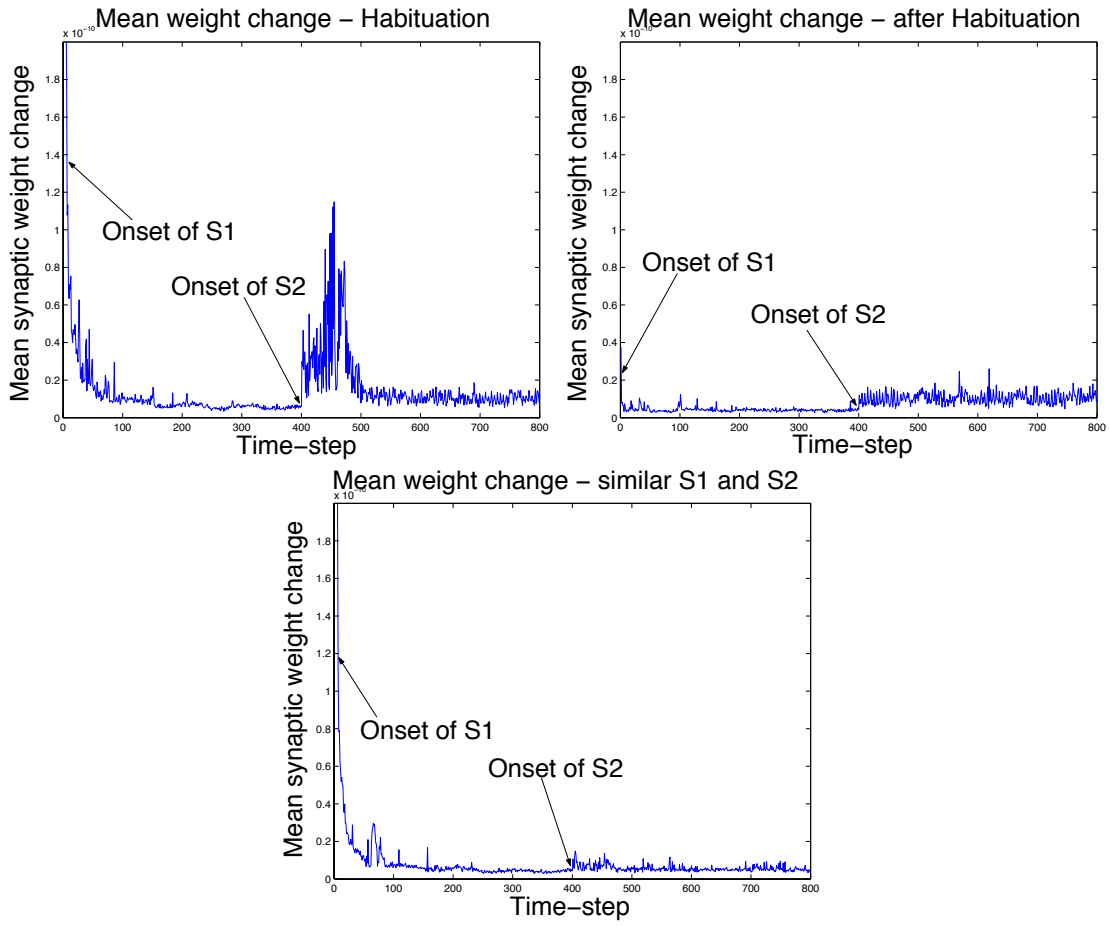


Figure 5-9: The top left panel shows the variation of the absolute mean weight change when the network is presented with S1 and then S2. The top right panel shows the mean weight change for the same network for S1 and then S2 after exposure to S1 and S2 in the top left panel. The lower panel is a plot of the mean weight change when the time constants of the frequency modulation of S1 and S2 are similar in value ($t_c = 2$ for S1 and $t_c = 2.5$ for S2).

connectivity.

So, the first result is that, given a recurrent network of spiking neurons, a subset of which receive a stochastic but time-invariant input regime, and in which all synapses of all neurons are subjected to $STDP + N$ learning, the information contained within the input regime becomes contained within the weight vectors of the network neurons. The observed apparent approximate convergence can be thought of as an indication of this learning. It should be noted that this apparent approximate convergence has not been shown formally. Rather, an indication that it exists has been shown experimentally. For the parameters detailed, convergence with Hebbian STDP learning occurred in every run observed.

An interesting result is that of the approximate equivalence of Hebbian and Anti-Hebbian learning, as shown in figure under specific conditions. In section 5.5.2 it was said that in order for Hebbian and Anti-Hebbian $STDP+N$ to be considered approximately equivalent, two conditions should be met. Firstly, a small learning rate ϕ should be used, and secondly, all synaptic weights should be approximately aligned with the spiking activity they receive, as indicated in figure 5-3.

The result of the approximate equivalence of Hebbian and Anti-Hebbian learning is also supported by the experiments in section 5.5.3. It was shown that if the network is exposed to the stimulus for a sufficiently large number of iterations of Hebbian learning, such that the mean synaptic weight change for the network has reached an approximate fixed point, then the subsequent application of Anti-Hebbian learning does not significantly change the specific weight vectors as much as if the number of iterations of Hebbian learning were less.

Figure 5-4 shows the changes that are made to the synaptic weights of a neuron exposed to 1000 iterations of Hebbian followed by 1000 iterations of Anti-Hebbian learning. It can be clearly seen that the individual synaptic weights that form the weight vector, undergo significant changes. Figure 5-5 shows the weight vector of a network neuron in which, the network was exposed to more Hebbian learning (5000 iterations) before the application of the Anti-Hebbian learning. It can be seen that in this case the weight vector is largely unchanged by the subsequent

1000 iterations of Anti-Hebbian learning.

Figure 5-6 shows two plots of the average synaptic weight change for each network neuron due to applying 1000 iterations of Anti-Hebbian learning to a network that has already been exposed to either 1000 or 5000 iterations of Hebbian learning with the same stimulus. These results are averaged over 10 networks for each of the two amounts of prior Hebbian learning. It can be seen that when the networks are exposed to 5000 iterations of Hebbian learning, the subsequent average synaptic weight change caused by an additional 1000 iterations of Anti-Hebbian learning is significantly smaller than in the case of only 1000 iterations of Hebbian learning. The mean change for 5000 iterations of Hebbian learning is $7.56e^{-10}$ with $SD = 2.73e^{-10}$, compared to $3.22e^{-09}$ with $SD = 7.13e^{-10}$ for 1000 iterations of Hebbian learning.

It appears that with the $STDP+N$ learning described here, once an approximate fixed point has been reached, it is likely to be exceedingly difficult for a network to have any significant synaptic changes effected on it by repeated learning — even when the subsequent learning is of a regime that is fundamentally 'opposite' to that to which it has been subjected. Longer periods of Hebbian learning on the weights of a stimulated network allow better alignment of the weights with the pre and post-synaptic firing activities of the target neuron. It should be noted that what is being discussed here is an *approximate* stable state/fixed point, and not an absolute fixed point, such as that which can be acheived with perceptron learning using the Perceptron Convergence Theorem, see Rosenblatt (1962), Haykin (1999), Duda *et al* (2001).

The Habituation experiments appear to show that it is possible, by applying the $STDP+N$ learning functions described in this chapter to a recurrent network of spiking LIF neurons, for a network to become habituated to a very specific spike train *structure* from a single spiking input neuron. Figure 5-9 shows that such a network then responds to subtle changes in the structure of the spike train on the *same* spiking input neuron by 'aligning' the synaptic weights with the new spiking activities. Additionally, it would appear that this happens without 'erasing' the weight changes which allowed for the habituation of the initial spike

train.

There is much more research to be done on this subject and future research could involve investigating the capabilities of such a learning approach in terms of the sensitivity to structural changes in the input stimuli itself and the number of stimuli that can be learned before the network begins to 'forget' previously learned stimuli. For example, one could apply input stimuli in which the precise timing of the spikes is important to the structure rather than simply the average frequency of the spikes. One could then investigate the effect of alternating between two distinct but precise input regimes on the Habituation of the network.

Chapter 6

A metric for time series of spikes

6.1 Introduction

The spiking neurons considered throughout the chapters of this dissertation communicate by emitting a temporally precise series of spikes, known as spike trains. In order to be able to perform an investigation in which it is necessary to know if one spike train is similar to another, or whether or not something has been learned, some sort of measure of spike train similarity is required.

A metric for weighted spike trains is introduced in this chapter which provides an accurate measure of distance between temporally precise spike trains. The set of time series of spikes is expanded into a vector space, V, by taking all linear combinations of spikes. A definition is then given for an inner product on this vector space. This gives a definition of norm, of distance between time series, and of orthogonality. This also allows us to compute the best approximation to a given time series which can be formed by a linear combination of some given collection of time series. It is shown how this can be applied to a very simple learning or approximation problem.

But what about the capacity of the single spiking neuron itself? It is shown in this chapter that a single neuron can be trained using the $STDP + N$ learning

from chapter 5, to produce a precise output spike train in response to a collection of specific input spike trains. To take this further, can a single neuron learn more than one precise input/output spike train association? If so how many, and how can it be increased? In an attempt to address these questions, this chapter contains an investigation of the capacity of a single neuron to learn multiple precise input/output spike train relationships or, I/O associations. The work contains an investigation into the relationship between the number of I/O associations that a single neuron can learn and the number of spike train inputs it receives. In this investigation, the metric established at the beginning of this chapter is utilised as the measure to establish how well a desired output goal spike train has been learnt.

It is shown that the number of unique, precise I/O associations that a single neuron can learn is highly dependent on the number of input connections that the neuron receives. A log-linear relationship exists between the number of inputs and the number of I/O associations that can be learned without erasing previously learned I/O associations.

6.2 The Metric

We define a spike at time t_1 to be a function $s(t_1)$ of time t so that $s(t_1)(t) = 1$ if $t = t_1$, and is zero otherwise. We define a time series to be a finite sum of spikes,

$$\sum_{i=1}^{N} s(t_i)$$

with $t_1, \ldots, t_N$ distinct times.

We define a weighted time series to be a finite sum of the form

$$\sum_{i=1}^{N} c_i s(t_i)$$

The coefficients c_i and the times t_i can be any real numbers, and the number of terms, N, can be any natural number. We let V be the vector space, over the

real numbers, of weighted time series, with the obvious definitions of addition and scalar multiplication. V is infinite dimensional with uncountable basis.

We consider the following basic problems. Suppose $w_1, \ldots, w_k$ are time series and suppose also that we are given a goal time series g, and an output neuron N, which behaves as one of the spiking models discussed, for example in Gerstner and Kistler (2002). Let $inp(N) = c_1 w_1 + \cdots + c_k w_k$ be input to N. Let $out(N)$ be the output time series produced by N when given this input.

- Problem 1). Find values of weights $c_1, \ldots, c_k$ so that $inp(N)$ is close to g.
- Problem 2). Find values of weights $c_1, \ldots, c_k$ so that $out(N)$ is close to g.

In order to say more precisely what "close" means, we define an inner product on V. A definition of closeness establishes a measure of how much of the goal output N has learned and is able to replicate in its output spiking pattern.

6.2.1 Inner Product

An inner product on a vector space, V, over the reals is a function
$< u, w >: V \times V \rightarrow R$ so that:

1. $< u, w > = < w, u >$.
2. $< u + v, w > = < u, w > + < v, w >$.
3. $< cu, w > = c < u, w >$, for any real c.
4. $< u, u > \geq 0$.
5. $< u, u > = 0$ only if $u = 0$.

Because of the linearity of the inner product, and since V is formed by linear combinations of spikes, we only need to define the inner product between two spikes. We define

$$< s(t_1), s(t_2) >= e^{-|t_1 - t_2|/\delta}$$

where δ is some scaling factor.

In general

$$< \sum c_i s(t_i), \sum d_j s(r_j) >= \sum c_i d_j e^{-|t_i - r_j|/\delta}$$

We should check that this is an inner product. Suppose $u = \sum c_i s(t_i)$. In order to show that $< u, u > \geq 0$, define $F(u)$ to be $\sum c_i e^{t_i - t} h(t_i)$, where $h(t_i)$ is the function of t which is equal to zero for $t < t_i$, and is equal to 1 for $t \geq t_i$. We may think of $F(u)$ as a hypothetical post synaptic response to weighted time series u. For simplicity, set time scale $\delta = 1$.

$$\int_{-\infty}^{\infty} F(s(t_1))F(s(t_2))dt = \int_{max(t_1,t_2)}^{\infty} e^{t_1 + t_2 - 2t} dt = 2 < s(t_1), s(t_2) >$$

In general $\int_{-\infty}^{\infty} F(u)F(v)dt = 2 < u, v >$. Since $\int_{-\infty}^{\infty} F(u)^2 dt = 0$ if and only if $u = 0$, we get conditions 4) and 5) above.

From an intuitive point of view $< u, v >$ measures correlation of u and v.

From this we get $norm(w) = \sqrt{(} < w, w >)$, $d(u, w) = norm(u - w)$, which gives us a metric on time series. Following the discussion above, we may think of $d(u, v)$ as a measure of the difference between hypothetical post synaptic responses to u and v. To give some idea of how this works, suppose $\delta = 1/33$, and time is measured in seconds. Then $d(s(t_1), s(t_1 + 0.01)) = 0.75$ approximately.

We also get u is orthogonal to w if and only if $< u, w > = 0$.

Additionally we get $Proj_w(u) = (< u, w > / < w, w >)w$. This is the projection of u onto direction w. This may be understood as the best approximation to u which can be expressed as a multiple of w.

Example 1. *Take time scale* $\delta = 1$. *Suppose* $w_1 = s(1) + s(2), w_2 = s(2) + s(3), w_3 = s(1) + s(3), u = s(2)$. *Then* $Proj_{w_1}(u) =< u, w_1 > / < w_1, w_1 > w_1 = s(1)/2 + s(2)/2$. *We note that, as expected,* $u - Proj_{w_1}(u)$ *is orthogonal to*

$Proj_{w_1}(u)$. We can use the Gram Schmidt process as usual to find an orthogonal basis for the subspace spanned by (w_1, w_2, w_3). Once we have this orthogonal basis, we can, as usual, find the best approximation in the subspace to any given element of V.

6.3 Approximation

We now get some solutions to problems 1 and 2.

6.3.1 Gram Schmidt Solution to Problem 1

Use Gram Schmidt process on $w_1, \ldots, w_k$ to get an orthogonal basis for the subspace $Span(w_1, \ldots, w_k)$. Suppose this orthogonal basis is $w_1*, ...w_m*$. We can find the best approximation to g in this subspace by

$$\sum proj_{w_i*}(g)$$

This is guaranteed to give the *optimal* solution to problem 1), i.e. the unique linear combination in the subspace which is closest to the goal.

6.3.2 Iterative Approximate Solution to Problem 1

This iterative method is less computationally intensive than the Gram Schmidt implementation just described. However, it does not produce the optimal approximation to the goal g. $w_1, \ldots, w_k$ will not be able to provide as accurate a representation of g as the orthogonal set $w_1*, ...w_m*$, but it is much easier to implement and produces a sufficiently accurate approximation.

$E = g - inp(N)$
WHILE ($norm(E)$ is large)
    Pick i at random
    $ch(c_i) := (Proj_{w_i} E)/(norm(w_i))$
    $c_i := c_i + ch(c_i)$
    $inp(N) := inp(N) + ch(c_i)w_i$
    $E = g - inp(N)$
END

Where E is the difference between the goal time series g and the input vector $inp(N)$.

6.3.3 Iterative Approximate Solution to Problem 2

Problem 1) was concerned with the application of the metric to show that the input vector $inp(N)$ could have its weights modified in such a way that it became close to the goal time-series g. Problem 2) is more practical in that the neuron N is now required to output the goal g. Instead of using $inp(N)$ to calculate E we now use $out(N)$ — the output time-series of N.

$E = g - out(N)$
WHILE ($norm(E)$ is large)
    Pick i at random
    $ch(c_i) := (Proj_{w_i} E)/(norm(w_i))$
    $c_i := c_i + ch(c_i)$
    $inp(N) := inp(N) + ch(c_i) w_i$
    $E = g - out(N)$
END

These iterative solutions are applied in section 6.4 and the results for problem 1) and problem 2) can be seen in figures 6-1 and 6-2 respectively.

6.4 Testing

The following tests were performed using the iterative algorithm, outlined in sections 6.3.2 and 6.3.3. The purpose of the first set of tests is to demonstrate the ability of the algorithm to alter weight values $c_1, ..., c_k$ such that *inp(N)* becomes close to a required goal time series, *g*. We are attempting to bring the distance — as defined by *norm(g-inp(N))* — between the goal and the input vector to a minimum.

For each experiment, the goal and each time series that make up *inp(N)* consist of 10 spikes that have been randomly drawn from a uniform distribution in the interval (0,1). The initial values of the weights $c_1, ..., c_k$ are set to zero. All experiments are performed in CSIM (IGI Group, 2008).

Figure 6-1 shows a plot of the distance between the goal and *inp(N)*, with respect to the number of iterations of the algorithm of section 6.3.2, where *inp(N)* consists of 500 input channels. In this experiment we used a time scale of 1/33; so, $< s(t_1), s(t_2) >= e^{-33|t_1 - t_2|}$. It can be clearly seen that initially, the distance falls sharply by a small amount before leveling off. The reason for this is simply

that any initial adjustment to the weight of an input channel is likely to have the effect of decreasing the distance by a relatively large amount.

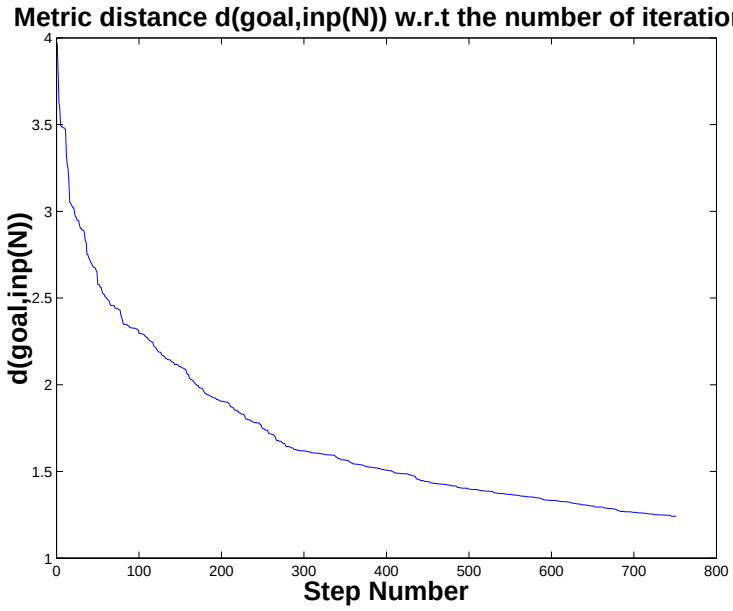


Figure 6-1: Distance between the goal and the input vector w.r.t step number, with *inp(N)* consisting of 500 spike trains.

Figure 6-1 clearly demonstrates that the iterative algorithm steadily reduces the distance between the desired goal and our input.The distance at step number 0 in figure 6-1 indicates the initial metric distance between the goal spike train and the output of the neuron before any training has occurred. As the weights are initially set to zero this distance is the metric distance between the goal spike train and an empty spike train.

The second set of experiments is designed to apply the iterative training algorithm to alter the input weights $c_1, ..., c_k$ of a spiking neuron which, receives *inp(N)* as an input, to produce our goal time series as an output.

The neuron used, N, is a basic Leaky Integrate and Fire (LIF) neuron, the parameters of which, can be found in the implementation appendix, A. The time scale of $1/33$ was used to match the time constant on which N operates. Similar

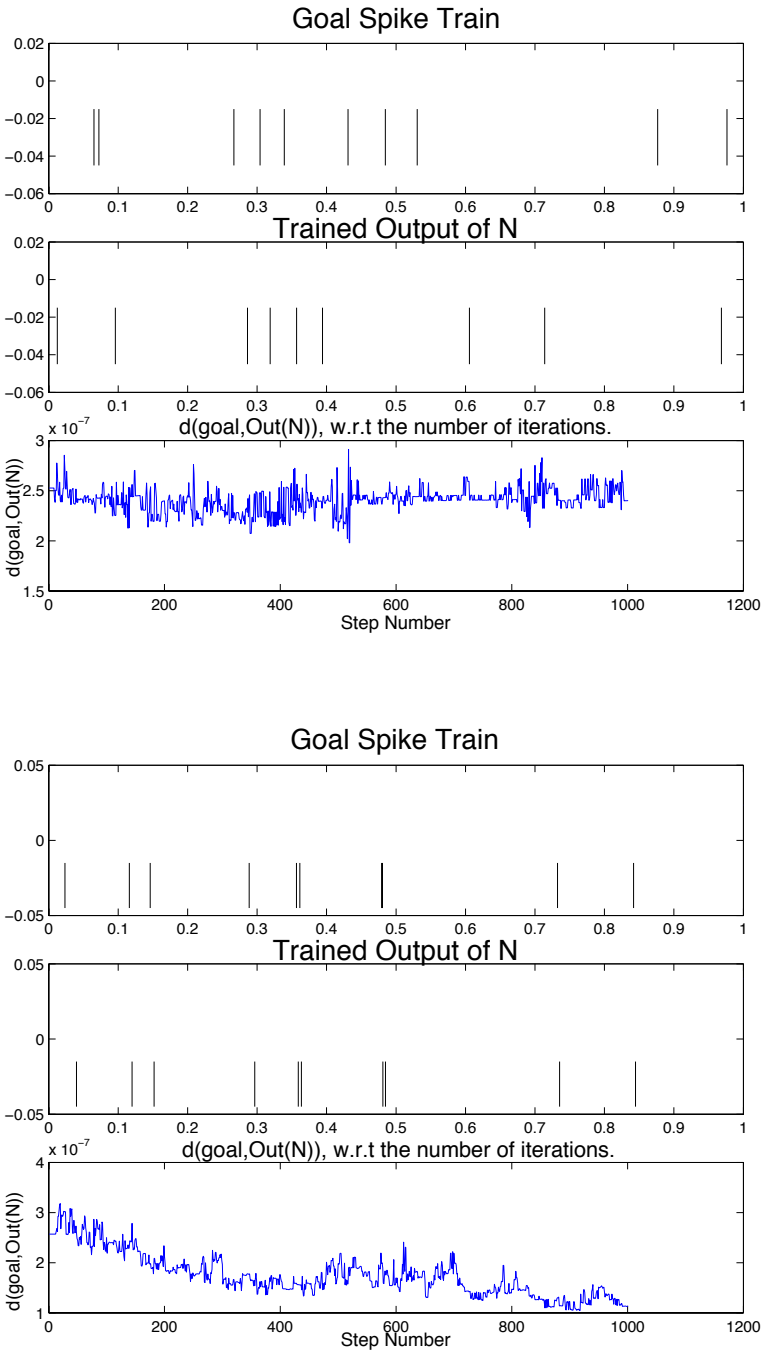


Figure 6-2: Randomly generated goal spike trains and the trained output produced by a LIF neuron and the associated distance between them during training with *inp(N)* consisting of 10 (upper panel) and 500 (lower panel) spike trains.

results were obtained with different time scales and parameters.

In the top segments both panels of figure 6-2 we can see our randomly generated goal time series and in the second segment is the spiking output of the LIF neuron — see Gerstner and Kistler (2002) — after training. The neuron output is now somewhat similar to our goal, but not very. This is linked to the fact that only 10 input spike trains were used.

The lower panel of figure 6-2 shows the result of increasing the number of input channels to 500. The much increased diversity of the spikes that populate *inp(N)* means that there is a much greater likelihood that we are able to construct our goal spike train with increased accuracy. The trained output is now extremely close to our goal.

The third segments of each of the panels in figure 6-2 illustrate the course of *norm(g-out(N))* with each iteration of the algorithm. This plot is noticeably different from the distance plots of figure 6-1. The peaks are due to the adjustment of weights which then cause the neuron to fire when it is not desired, or to lose a spike where it is desired. This over adjustment is then corrected by a single sharp change or series of sharp changes. For the 500 channel case, it can be seen that the general trend is that the distance decreases as the number of iterations increases. The distance plot for the 10 channel case shows very little decrease for the same number of iterations.

It is clear that to construct the input to a spiking neuron in order to produce an accurate representation of a specific goal time series it is necessary that the input vectors be highly diverse.

6.5 Neuron Type and Task Suitability

The results just presented use a slightly modified LIF model to that described in the implementation section. The difference is in the way the internal state of the neuron is reset after firing. In the standard LIF model described in the

implementation section, the internal state was reset to a value well below the threshold value and below the resting potential value. However it was found that using such a model placed too much limitation on the diversity of the spike train the neuron could perform. The time needed for the internal state of the neuron to rise from its very low reset value to a value at which spiking was again likely was too long a period — long enough that a second firing in close temporal proximity to a previous firing would not be possible.

There are numerous ways to address this problem. It was decided that the internal state reset value would be increased to be above the resting value but below the firing threshold value. This change means that the neuron is now able to perform faster successive and precise firing. This modified model will be referred to as the fast spiking model. The time constant of the internal state of the neuron was not altered as this would adversely affect the duration that a post synaptic spike remained in the internal state of the neuron and would therefore limit the number of post synaptic spikes that would be able to interact with each other within the internal state of a LIF neuron.

Figure 6-3 shows the values, each averaged over 40 runs, of $d(goal, actual)$ for the standard and fast spiking models, along with the 95% confidence intervals. It can be seen that there is a slight trend for better performance from the fast spiking neuron compared to the slow spiking neuron. Based on this trend, the fast spiking neuron was used for this chapter. See table A.1 in section A.1.1 for exact values of the reset values.

6.6 Discussion of the Metric $d(u, v)$

One of the good properties of this metric is that it is continuous with respect to variation in times of spikes, as well variation of coefficients. We have $lim_{\epsilon \to 0}$ $d(s(t_1), s(t_2 + \epsilon)) = d(s(t_1), s(t_2))$. Also any two weighted time series u and v are connected by the straight line $(1 - x)u + xv$, as x goes from 0 to 1.

This should be contrasted with the more usual approach, which is to divide a time

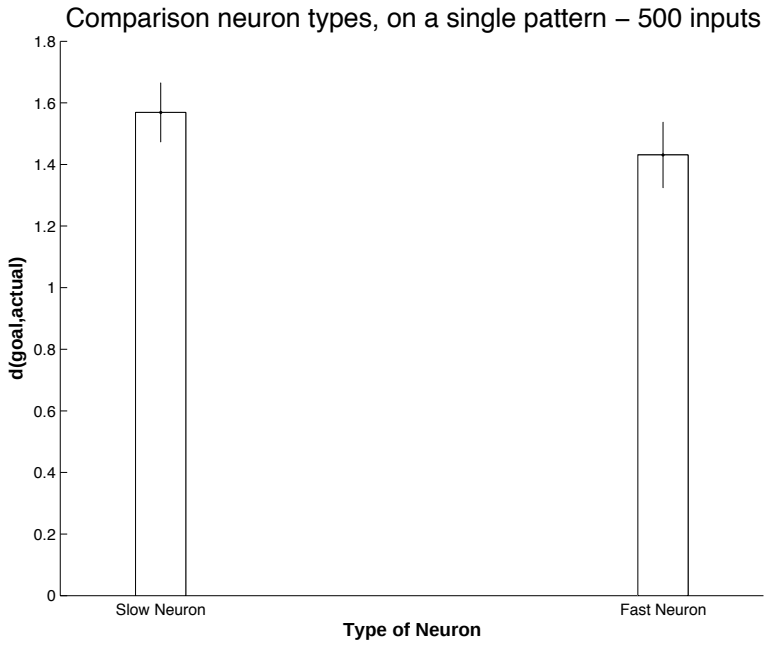


Figure 6-3: A plot of $d(goal, actual)$ values for *fast* and *slow* neuron types Each has been averaged over 40 uniquely generated patterns.

interval into small subintervals, and to represent a time series of spikes by a vector of zeroes and ones, the length of the vector being the number of subintervals, and the ith component of the vector being 1 if and only if a spike occurs in the ith sub-interval. We can then define some metric on these Boolean vectors, but however this is done it will not vary continuously with small perturbations in the times of the spikes. Also it is not always clear what would be a good path from one Boolean vector to another, especially if they have different numbers of spikes.

Another good feature of our metric is that it gives a unique optimal solution to Problem 1), whereas if Boolean vectors are used, the result depends on the size of the subintervals.

6.7 Related Metrics

This metric is not entirely new, but is similar, for example, to the metric introduced in van Rossum (2001). As in the metric described here, the van Rossum metric replaces the delta function of a spike with an exponential function and uses the ℓ^2-norm of the difference between two spike trains to provide a notion of the distance between the two spike trains. While there are great similarities between these two metrics there are some differences. For the metric described in this thesis, the vector space, V, is defined for *weighted* spike trains. Each spike has a weight associated with it. To obtain a measure of *closeness* of two spike trains, an inner product was defined, and confirmed, for V.

Another similar metric is described in Victor and Purpura (1996), on which the previously mentioned metric of van Rossum (2001) is based. Victor and Purpura allow for the transformation of one spike train into another by the minimising of a cost function. Addition and deletion of a spike are assigned a cost of 1, while temporal shifting of a spike is assigned an arbitrary value that is proportional to the amount of time the spike is shifted. Therefore, the distance between any two spike trains is given by calculating the minimal cost function between the two spike trains. Other, more complex metrics have been devised that are based on the Victor and Purpura metric.

Another similarity between the metric of this thesis, and the metrics of van Rossum (2001) and of Victor and Purpura (1996), is that they all allow for a non-binned approach, and thus provide a continuous measure over the entire spike train and therefore, are better measures of similarity between spike trains than the typical 'binned' approach.

In the metrics outlined in van Rossum (2001) and in Victor and Purpura (1996), during the transformation of one spike train to another, a spike can be inserted, deleted or shifted in time. An advantage of the metric outlined in this thesis is that each spike is weighted and not simply 'there' or 'not there'. Spike trains can exist at varying strengths, which are determined by the synaptic weight on the connection from which the spike train originates. This use of a weighted spike

train means that the metric of this thesis can be successfully applied to solve the previously described problem 2) as in section 6.3.3. Spike train reconstruction was hinted at in van Rossum (2001) but not actually demonstrated.

6.8 Usage of STDP to obtain precise spike trains

The metric outlined in this chapter has been shown to be highly capable of modifying the synaptic weights of a spiking LIF neuron in order for the neuron to output a precise spike train in response to a specific and highly diverse collection of input spike trains, see figures 6-1 and 6-2.

However, one might say that given that the metric has access to the error function between the goal and the actual output spike train, in addition to being provided with a highly diverse input, the success of the metric is to be expected.

In an effort to accomplish the same problem in a more biologically realistic manner, the metric shall be replaced with the $STDP+N$ learning described in detail in chapter 5 of this thesis, and it is used here with some changes to certain parameters. These changes, along with explanations for each, shall be discussed in due course.

Consider the same experimental setup as previously described and illustrated in figure 6-4 below. The 500 inputs each generate 10 spikes that are distributed randomly throughout the interval $(0, 1)s$. Each of these inputs are connected to an *output* LIF neuron by 500 synapses. A goal spike train is generated which consists of 5 spikes. Each spike of the goal spike train is randomly generated also in the interval $(0, 1)s$. The choice of an average frequency of $10Hz$ for the input neurons, and $5Hz$ for the output neuron can be explained as follows. Firstly, it is important to realise that both frequencies are entirely feasible working frequencies for neurons of the human neocortex (Ojemann and Schoenfield-McNeill, 1998). Direct measurement of the activity of individual neurons *in vivo* indicates that the average firing rate of some neurons can be $3Hz$ in highly isolated neurons, with maximal frequencies being $11Hz$ (Ojemann and Schoenfield-McNeill,

1998). It is not suggested here that, these frequencies should be treated as being absolutely correct, but they do provide a meaningful basis on which to construct the spike trains used here. It is assumed that a neuron is able to more accurately construct a target output spike train when given a collection of input spike trains, when the input spike trains consist of many spikes compared to the number of spikes contained in the target output spike train. Therefore, with this assumption in mind, the average frequency of the output spike train was set at $5Hz$ — comparable to actual frequencies observed in Ojemann and Schoenfield-McNeill (1998) in highly isolated neurons, while the average frequency of the input neurons was set to $10Hz$ — close to the observed maximal firing frequency of individual neurons observed by Ojemann and Schoenfield-McNeill (1998). As a result of these choices, the target neuron is given one of the best possible chances of learning its goal spike train, while maintaining realism. It should be noted that both of these firing rates are much lower than the firing rates that are typically used in simulating spiking neurons. For example the work Legenstein *et al* (2005), much higher frequencies of $25Hz$ are used. While it is possible for such firing rates to occur naturally in populations of neurons, observational evidence obtained by Ojemann and Schoenfield-McNeill (1998) would seem to indicate that these frequencies are not necessarily typical at the level of the individual neuron.

Before continuing with the experiments the *norm* of the weight vector to the target neuron needs to be set to an appropriate value. Consider a target LIF neuron with 500 input channels, the input synaptic weights of this neuron are trained using $STDP + N$ learning technique so that the post synaptic spike train of the target neuron, referred to as the actual output, is close to the goal spike train as determined by the distance, $d(goal, actual)$, using the previously described metric.The norm of the input vector was chosen to be 9×10^{-9} as this gave an average number of output spikes very close to the number of spikes in the goal spike train i.e. 5 spikes.

In order to maintain realism and, in a departure from the approach used when using the inner product learning rule, the sign of all of the input synapses is fixed and cannot be changed by the learning rule. This is accomplished using the

following procedure: Suppose that the modification, calculated by the learning rule, that is to be applied to a synaptic weight of magnitude w is of a magnitude greater than w and is also negative. The weight must not be allowed to become negative, so instead it is set equal to zero. It should be noted that the use of this approach, combined with too large a learning rate will cause virtually *any* negative change to be large enough to cause the relevant synaptic weight to be set to zero. Therefore, a value for the learning rate must be chosen that is large enough for learning to occur within a reasonable time-frame, but not so large that it causes a great number of synapses to be set to zero.

The disadvantage of this approach is that some performance will be sacrificed in the form of the similarity that can be achieved between the goal and the actual spike train after training. All of the synaptic weights used are positive. This is because it is unclear how to treat inhibitory synapses using the STDP learning rule. For example, with excitatory synapses, if a pre-synaptic neuron fires before the the post-synaptic neuron it is treated as though it may have contributed to the post synaptic firing and is strengthened. If it fires before the post-synaptic neuron, it is weakened. Now consider an inhibitory synapse, if the pre-synaptic neuron fires before the post-synaptic neuron it cannot necessarily be considered to have contributed to the post-synaptic neuron firing, due to its inhibitory effect — though it could perhaps be thought of as affecting the timing of the post-synaptic neuron firing, in terms of delaying it. This fundamental difference means that it is difficult to reconcile using standard STDP learning on inhibitory connections. Indeed, it may be the case that inhibitory neurons should be treated by an altogether different learning regime to STDP, such as a homeostatic learning rule similar to that used in Moldakarimov *et al* (2006). A homeostatic learning rule is applied to a purpose-built neural network in which an excitatory neuron receives an input from an inhibitory neuron and also receives an input from itself (self excitation). The inhibitory neuron receives an input from the excitatory. The synaptic weight from the inhibitory neuron to the excitatory neuron is increased if the firing rate of the excitatory neuron rises above some threshold value and is decreased if the firing rate of the excitatory neuron falls below the threshold value. The use of a homeostatic learning rule on inhibitory

synapses acts as a regulating influence on the firing activity of the excitatory neuron that ensures that its firing frequency does not rise too high, or drop too low.

While the majority of research has involved STDP learning with excitatory synapses, there has been a relative lack of work done on plasticity within inhibitory synapses. Some evidence (Haas *et al*, 2006) has been published that shows that a form of STDP learning operates on inhibitory synapses of the entorhinal cortex. Haas *et al* found that the strength of synaptic modification varied as a function of $\Delta t = t_{post} - t_{pre}$ where t_{post} and t_{pre} are the firing times of the post-synaptic and pre-synaptic neurons respectively. It was found that potentiation of the inhibitory synapses occurred for $\Delta t > 0$ and that depression occurred for $\Delta t < 0$.

However, a review of some of the research into plasticity mechanisms for inhibitory neurons (Gaiarsa *et al*, 2002) highlights that mechanisms for inhibitory synaptic modification vary throughout different brain regions.

The focus of the research in this chapter is aimed at understanding how much a neuron can learn. The introduction of plasticity on inhibitory synapses would introduce another layer of complication and would in itself be worthy of further research as an extension to the research established here. This future research could aim to examine the effect of inhibitory synaptic plasticity on what a neuron can learn, compared to the results obtained here for excitatory plasticity only.

An additional point for consideration is that the use of the normalisation procedure outlined in chapter 5, can be thought of as effecting the stabilisation that homeostatic learning with inhibitory synapses would provide so, at least in this respect, the inhibitory synapses are not required for this implementation.

6.9 STDP Precise Spike Train Results

Consider the experimental setup just described and the previously described $STDP + N$ leaning technique to modify the input synapses. During a single training iteration for a particular Input/Output association, the input of 500 randomly generated input spike trains is presented to the output neuron. The output neuron has its output spike train clamped to perform the desired goal spike train during the presentation of the input. The term *clamped* means that the neuron is forced by an external stimulus, to perform a desired precise firing sequence. For example, the stimulus could be a neuron with a strong synaptic connection to the output neuron. The $STDP + N$ learning is then applied to each of the input synapses using the appropriate input spike train along with the clamped output spike train of the output neuron. This training is repeated for 50 iterations.

These initial results confirm that the $STDP + N$ learning outlined here, when applied to the input synapses of the output neuron is indeed capable of producing precisely timed output spike trains from the output neuron. This results is similar to that obtained in Legenstein *et al* (2005), and the work was obtained independently and concurrently with their result.

As well as showing that $STDP + N$ learning can indeed be used to teach a single neuron to perform a precise spike train, figure 6-5 shows examples of both good and not so good metric distances. These are provided so that it can be seen what a particular metric distance value between a goal and an actual spike train actually means.

6.10 Multiple Pattern Training Techniques

Given the previous result of $STDP + N$ Hebbian learning enabling a neuron to produce a precise spike train in response to a specific input pattern, it is possible to expand upon this result and investigate the capability of a neuron to learn

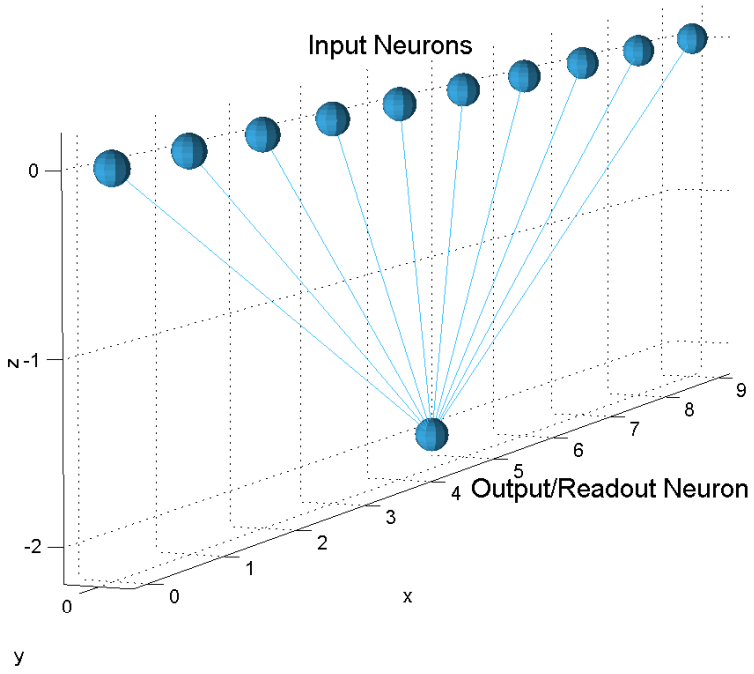


Figure 6-4: An illustration showing a single output/readout neuron that has 10 input neurons connected to it. This is a rudimentary setup as the actual setup involves the use of many hundreds or even thousands of input neurons — only 10 are shown here for illustrative purposes.

multiple, precise input output associations. The weight vector resulting from a neuron being trained to produce a precise output is a diverse combination of many values, in this case 500. In order for multiple I/O associations to be learnt by the same neuron, will obviously require the existence of some sort of *compromise*, in which the individual weight of the weight vector are optimised for no pattern in particular. If this were to be possible one could suppose that the precision of the learned output spike train must be reasonably robust to changes within the trained weight vector — perhaps relying on just a small proportion of the individual elements of the vector.

Consider the same experimental setup used previously, and illustrated conceptually in figure 6-6. In order to investigate the ability of a single neuron to learn multiple I/O associations, it will be necessary to first define the training tech-

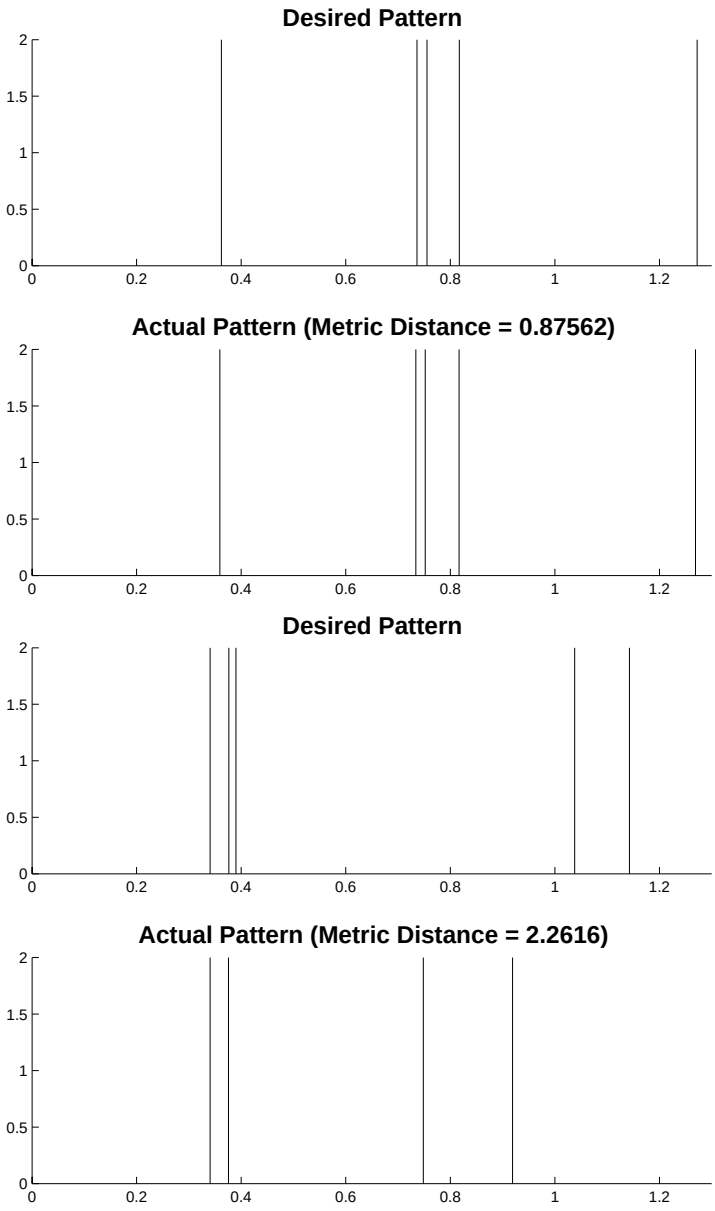


Figure 6-5: These two figures are examples of two specific goal spike trains and the actual output spike train of two separate single neurons after training. The upper figure is of an output spike train that is highly similar to the goal with a metric distance of less than 1. The lower figure shows a less successful example in which, after training, the output spike train of the single neuron has a metric distance from the goal of greater than 2.

niques that will be used. For the purposes of the following experiments, the training technique refers to the method of presentation of the multiple I/O associations to the target neuron. For the sake of brevity, the number of patterns used throughout the description of the techniques is limited to 2.

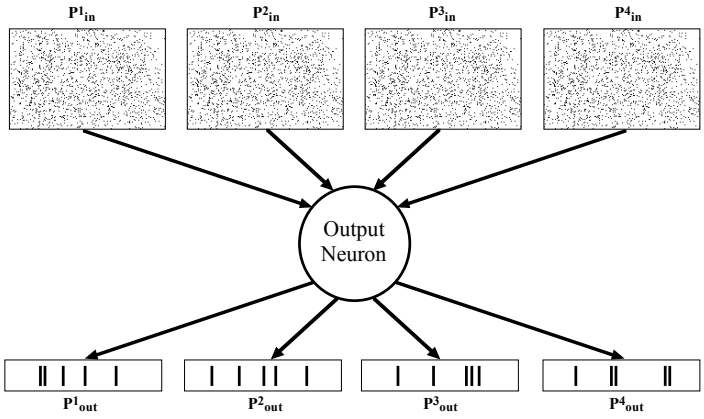


Figure 6-6: A conceptual illustration of using a single neuron to learn multiple I/O associations simultaneously. Given input pattern P^1_{in} the output neuron will produce, as best it can, output pattern P^1_{out}, and similarly for the second, third and fourth input and output patterns.

6.10.1 Serial Pattern Presentation

The first, most basic technique will be referred to as *serial* learning. Using this method, the first input pattern P^1_{in}- comprising of 500 spike trains, each containing 10 spikes over a 1s interval — is presented to the target neuron. During the same 1s interval i.e while it is receiving the input pattern, the target neuron is clamped such that it will perform the goal precise spike train output pattern P^1_{out} which it is required to learn as a response P^1_{in}. The synaptic weight changes are calculated according the $STDP + N$ learning regime discussed previously. The target neuron is exposed to 50 iterations of this pattern presentation, clamping and learning. After 50 iterations, the input pattern is changed to a second pattern

P^2_{in} that is randomly generated and is different to the first input pattern. The output pattern is also changed to pattern P^2_{out} that is distinct from pattern P^1_{out}. The target neuron is then subjected to a further 50 iterations of learning with the second I/O pattern association. For a target neuron N, the procedure can be written more formally as follows:

```
FOR (i=1 To 2)
    FOR (j=1 TO 50)
        Reset Network
        FOR ( simulation time of 1.3s )
            IF ( simulation time >0.3s )
                Present Input Pattern P^i_in;
                Clamp N to perform output pattern P^i_out;
                Calculate and apply weight vector update
            END
        END
    END
END
```

The network is simulated for $1.3s$, but input activity, clamping and therefore learning do not start until $0.3s$ into the simulation. This was to allow the initial internal state of the neuron to fall to resting values before commencing the training. Learning, therefore, is over a period of $1s$. Simulation means that the spiking activity and learning are active and that the synaptic weight changes due to pre- and post- synaptic firing activities are being calculated and applied.

6.10.2 Alternate Pattern Presentation

Using the *alternate* learning technique, the I/O associations P^1_{in} with P^1_{out} and P^2_{in} with P^2_{out} are presented at alternating iterations, for a total number of iterations that is equal to the product of the number of associations and 50, so in this

case for 100 iterations. For a target neuron N that is to be trained on two I/O associations, this can be more formally described as:

```
i=1;
FOR (j=1 TO 100)
      Reset Network;
      FOR ( simulation time of 1.3s )
            IF ( simulation time >0.3s )
               Present Input Pattern P_in^i;
               Clamp N to perform output pattern P_out^i;
               Calculate and apply weight vector update;
            END
      END
      IF (i==1)
         i=2;
      ELSEIF (i==2)
         i=1;
      END
END
```

6.10.3 Superimposed Pattern Presentation

Superimposed learning is a variation on the Alternate technique just described, with the difference being that the synaptic weights are not updated after each iteration. Instead, each I/O association is presented iteratively for the appropriate number of iterations. So, if it is required that n I/O associations be learnt then after each I/O presentation the weight update is calculated and stored, but not applied, except after each presentation of the n^{th} I/O association, at which point the weight vector is updated with the average of the weight vector updates for the last n iterations.

Procedurally, this looks like:

```
i=1;
FOR (j=1 TO 100)
      Reset Network;
      FOR ( simulation time of 1.3s )
            IF (simulation time >0.3s )
               Present Input Pattern P_in^i;
               Clamp N to perform output pattern P_out^i;
               Calculate and store weight vector update as δW_i;
            END
      END
      IF (i==1)
         i=2;
      ELSEIF (i==2)
         Update Vector = Average vector for all δW_i;
         Use Update Vector to modify the weight vector;
         i=1;
      END
END
```

In the following experiments, different numbers of input neurons to the single neuron are used. Altering the number of inputs means that the total synaptic contribution to the neuron is altered, and therefore the spiking activity of the neuron is altered too. In order to preserve the total synaptic input and make it a constant, such that the effect of the training can be compared as the number of inputs changes, the norm is changed, and its value depends on the number of inputs. A norm of 9.0×10^{-9} was chosen for the 500 input case. This value provided an average spiking output very close to the desired goal output of 5 spikes. A doubling of the number of inputs to 1000, means that the norm — such as it was previously described — must be multiplied by $\frac{1}{\sqrt{2}}$, resulting in a norm of $\frac{9.0\times10^{-9}}{\sqrt{2}}$ for 1000 inputs. This now means that during training, the norms of

the input vector to the single neuron, are set such that the total synaptic input remains constant as number of inputs increases. The training norms for 2000 and 4000 inputs are $\frac{9.0\times10^{-9}}{(\sqrt{2})^2}$ and $\frac{9.0\times10^{-9}}{(\sqrt{2})^3}$ respectively, using the same method.

Additionally, in order to maintain consistency in training while investigating the effect of changing the number of inputs, the learning rate — which determines the strength of the absolute change to a synaptic weight before normalising takes place — is also altered as the number of inputs increases. Consider that as the number of inputs increases, and the absolute size of individual synaptic weights decreases, then in order to maintain the *strength* of the learning, the absolute change in a synaptic weight must also decrease. This is accomplished by ensuring that there exists an inverse relationship between the learning rate and the number of inputs, such that a doubling in the number of inputs results in the learning rate being halved. If this does not happen, and the same learning rate that was used for 500 inputs, is also used for 4000 inputs, then the relative change in the 4000 input case will be much greater than in the 500 case. The learning experienced by the synapses of the single neuron will be much higher and the results for determining the effect of increasing the number of inputs, would be flawed.

6.10.4 Multiple Pattern Learning Results and Analysis

Figure 6-7 shows a plot of the mean values of $d(goal, actual)$, the distance between the desired goal spike train for the target neuron and its actual output spike train after training for the different training techniques just described and for cases of 1, 2, 3 and 4 I/O associations, over 20 runs in each case. For those instances in figure 6-7 that show the result for multiple patterns, $d(goal, actual)$ is given for each pattern. The idea is to train a readout neuron on multiple I/O associations and observe the accuracy of the output of the trained neuron for each pattern it is required to learn, as the number of patterns increases.

Along with each measure of mean distance, the 95% confidence interval is also provided so that significant differences between the results of using different training techniques and/or different numbers of I/O associations can be clearly seen.

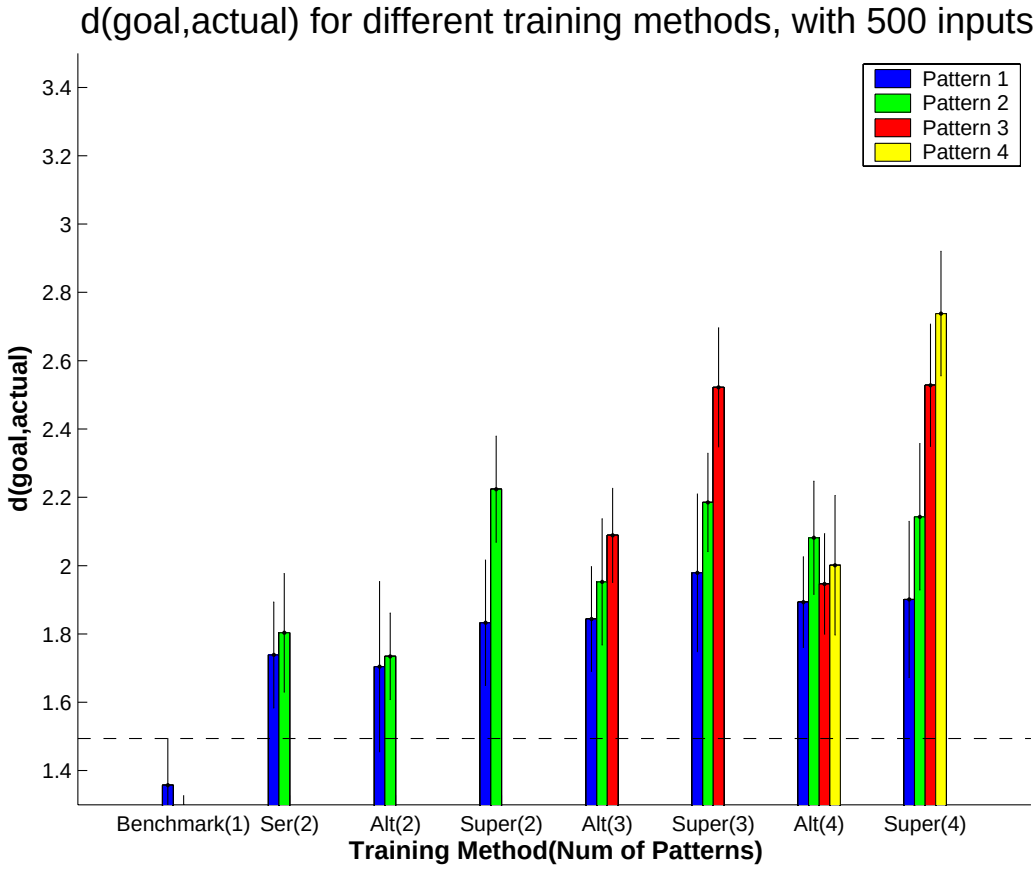


Figure 6-7: A plot of the mean values of $d(goal, actual)$ for varying number of patterns and training methods. Averaged over 20 runs in each scenario.

For a given training method and number of I/O associations, we can say with 95% confidence level that the given confidence interval will contain the value of $d(goal, actual)$ obtained from a single run. The difference between any two mean values of $d(goal, actual)$ can be considered significant (at the 95% level) if their confidence intervals do not overlap.

The distances are calculated using the metric described previously in this chapter. The metric acts as an objective measure of how much of the goal pattern the neuron has learnt, or how well the neuron has learnt a particular goal pattern. In the experiments that follow, there are many examples of the readout neuron learning some patterns well, and some not so well, but in all cases the learning is

imperfect. The metric is ideal as a measure of learning in such cases as it allows the amount of learning a neuron has undergone to be quantified, and as a result comparisons can be made between how much a neuron has learnt of two or more patterns.

It should be noted that in addition to changing the learning rate and the norm to maintain the relative strength of the training throughout changes in network size, another check is also implemented. It was found in preliminary testing that as the number of patterns, on which a neuron was trained, was increased, the number of spikes contained in the resulting output spike trains for each pattern of the trained neuron, also increased. The increase was such that it was possible that the observed poor performance — high $d(goal, actual)$ values — could have been attributed to too many spikes in the outputs, something which could perhaps be remedied by reducing the norm of the input weight vector to maintain the average number of output spikes close to 5, over all of the patterns on which the neuron was trained. As in figure 6-7, it can be seen that the greater the number of patterns a neuron is required to learn, the less well it is able to learn them, even with the output maintained at the goal of 5 spikes.

For each of the training technique/number of I/O associations shown, a total of 20 networks were randomly generated, and the relevant distances calculated for each. The average was then recorded and plotted in figure 6-7.

Figure 6-8 shows the number of spikes present in the actual output of the target neuron after training for each of the combinations of training technique/number of I/O associations used. It shows that the average number of spikes in the output spike trains of the neuron, over all patterns learned in each instance, is kept close to 5.

It is thought that the need to change the norm as the number of patterns increases arise because, as more patterns are required to be learnt, more synaptic weight are strengthened. This results in a greater synaptic input to the neuron and therefore a lower norm must used to maintain spiking frequency.

The first bar in figure 6-7 shows the average distance between a goal spike train

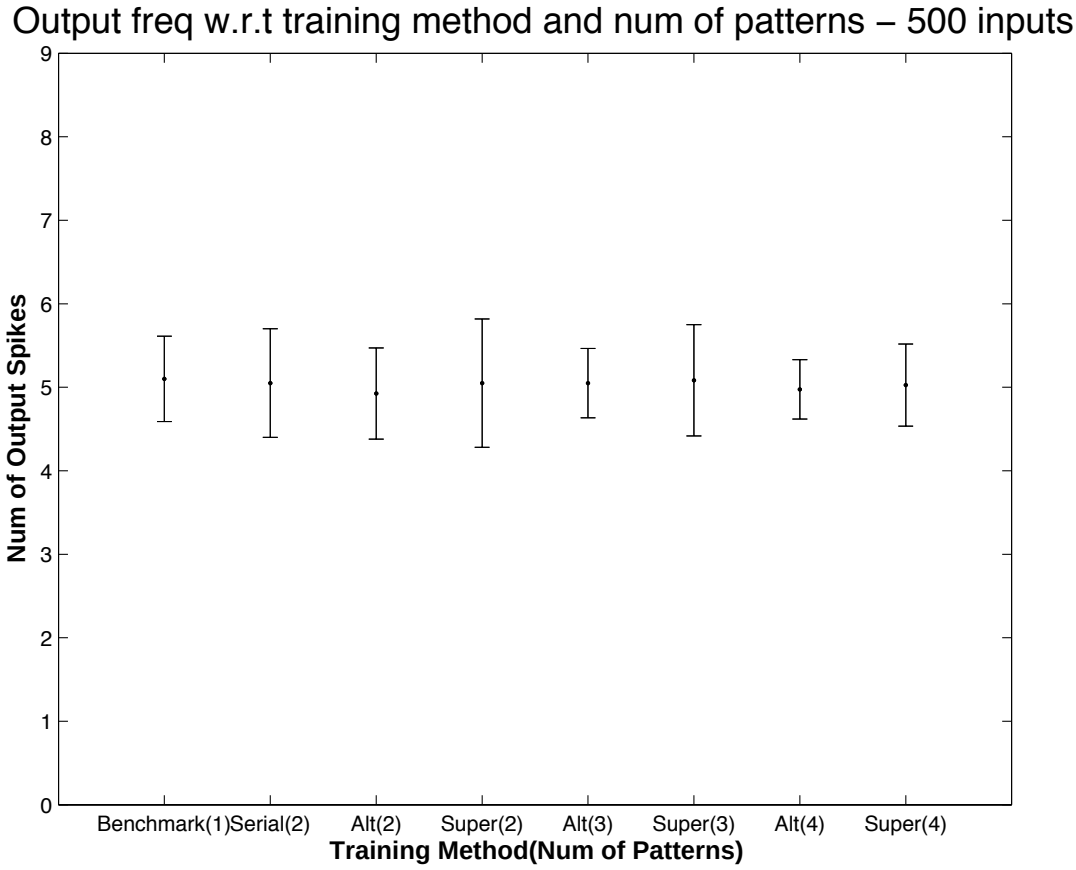


Figure 6-8: The average number of spikes contained in the output spike train of the output neuron over the 20 runs for each training method-number of patterns combination used. Also shown are the 95% confidence intervals for each training method.

and a precise output spike train produced by the $STDP + N$ learning rule when the 500 input synapses to a single neuron, are trained over 50 repetitions, on a single I/O pattern and averaged over 20 unique, randomly generated I/O patterns. This is the benchmark distance result, against which all other results shall be compared.

It can be seen that some entries in figure 6-7 have up to 4 differently coloured bars. Each of these bars represent a distance $d(goal, actual)$ for one of the goal spike trains on which the target neuron has been trained.

The first instance of training the target neuron to learn two patterns uses the serial method, discussed above. The serial technique was mentioned here for completeness. In reality, it is not a viable training technique due to the fact that, as the number of training repetitions increases, the previous learnt patterns get effectively *erased*. Therefore, all other results that can be seen in figure 6-7 for number of I/O associations 2, 3 and 4 are obtained using only the super and alternate techniques.

The next two results both also represent attempts to train a neuron to learn 2 precise I/O associations, using the alternate and the superposition techniques. It can be seen that both techniques are capable of producing weight vectors that allow for the weight vector of the target neuron to store two I/O associations that are not quite of comparable accuracy to the benchmark single pattern.

As the number of I/O associations increase to 3 and 4, it can be seen that the results for $d(goal, actual)$ for each of the output spike trains of the target neuron, increase so much that they are no longer comparable to the benchmark — their confidence intervals no longer overlap. It can also be seen that the alternate technique consistently outperforms the superimposed technique. Additionally, it would appear that increasing the number of patterns a single neuron is required to *know* at any time, results in a decrease in the accuracy and the integrity of all of the patterns learnt.

6.11 Increasing Number of Inputs

It has been shown that a target neuron with 500 inputs and having undergone the training described, cannot adequately learn 4 multiple patterns. A pattern is considered to be 'adequately learnt', if the confidence interval of the average $d(goal, actual)$ value — taken over 20 different I/O associations — overlaps with the confidence interval of the benchmark case, i.e. is not obviously worse than the worst results of the benchmark case. Figure 6-9 shows the effect on $d(goal, actual)$ of using different numbers of input channels — $500, 1000$, 2000 and 4000 — whilst

keeping the number of patterns equal to 4. Also shown is the result of using these different numbers of inputs to learn a single I/O pattern.

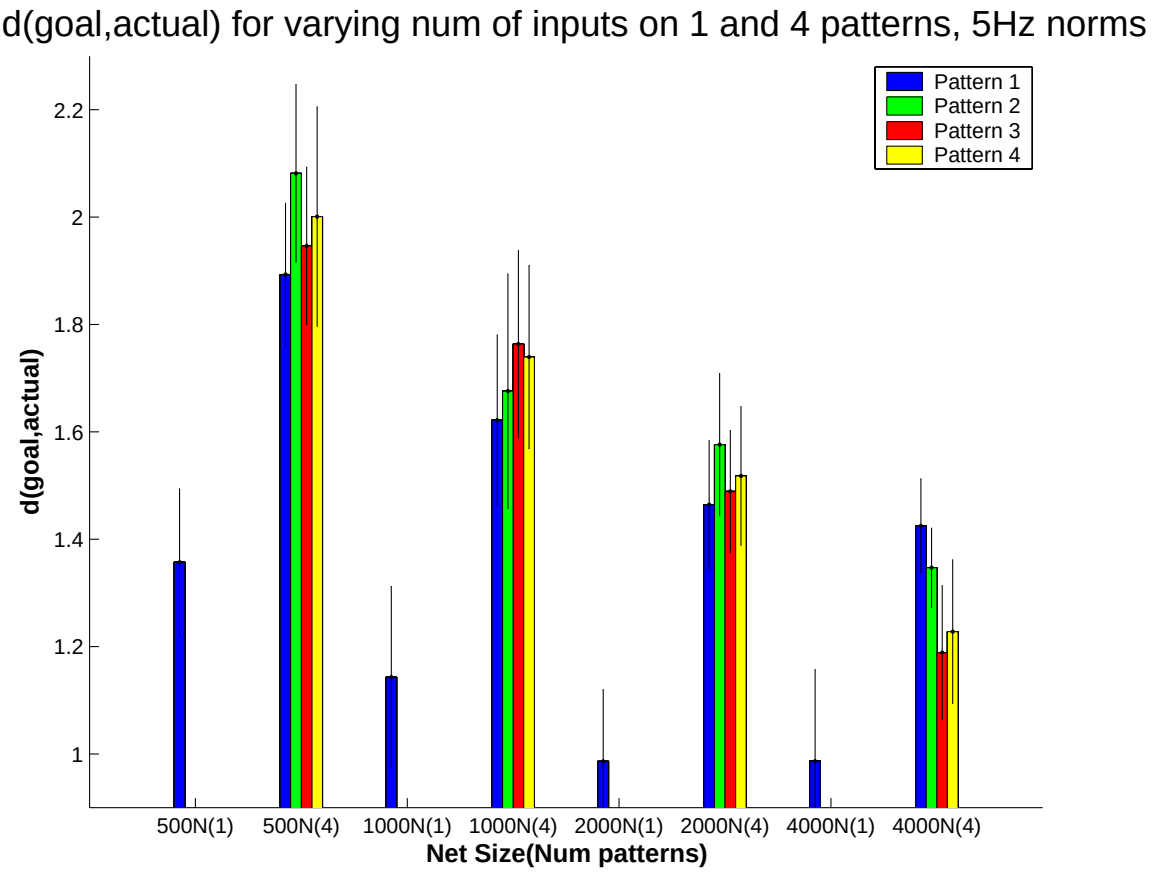


Figure 6-9: This figure shows the plots of the average values of $d(goal, actual)$ for varying numbers of input connections to a single neuron, along with the $d(goal, actual)$ values for each size network on learning a single pattern.

In the case of using 500 neurons, it can be seen that the average distances from the goal for each of the 4 patterns fall far outside the confidence interval of the benchmark case. Figure 6-9 shows the goal and the actual spike train of one such neuron. It shows a limited number of spike correlations between the actual output spike train and the goal spike train. In figure 6-9 it can be seen that as the number input channels increases, the average values over 20 runs, of $d(goal, actual)$ for each of the four patterns decreases. In fact the distance decreases by so much

that in the 4000 input neuron case, the distance is comparable to the benchmark distance for a neuron with 500 inputs learning a single pattern. It can also be seen that more inputs allow for single I/O pattern to be learnt that have lower $d(goal, actual)$ values than for the case of fewer inputs.

6.12 More Patterns

The purpose of this section is to increase the number of patterns that single neuron with a given number of inputs is trained on, until a point is reached where the patterns are not able to be all stored successfully.

The dotted line in each of the figures 6-10, 6-11, 6-12 and 6-13 represents the absolute upper limit of the confidence interval of the average $d(goal, actual)$ value for the 500 input benchmark case, in which the a single neuron with 500 input neurons was trained on a single I/O association. This line is included to better show which patterns do not have an overlapping confidence interval with the benchmark. A pattern is said to be learnt successfully if its confidence interval overlaps with that of the 500 input single pattern benchmark case, or if it is significantly less than that of the benchmark.

The preceding figures show that the 500 input single neuron is only capable of learning a single I/O association, as the other collections of numbers of patterns have at least one pattern whose confidence interval does not overlap with that of the benchmark. The 1000 input case is able to learn at least a maximum number of 3 I/O associations before integrity is lost and confidence intervals cease to overlap. Similarly the 2000 input case can learn at least a maximum of 4 I/O associations, while the 4000 input case is capable of learning at least a maximum of 6 I/O associations. Additionally, the 4000 input case is also able to learn 4 patterns with most of the patterns being more accurate than the benchmark case.

Figure 6-14 shows examples of the input, goal and actual spike trains for a single neuron with 4000 input spike trains. These specific examples are indicative that it is possible for a single neuron to learn a maximum of at least 6 patterns simul-

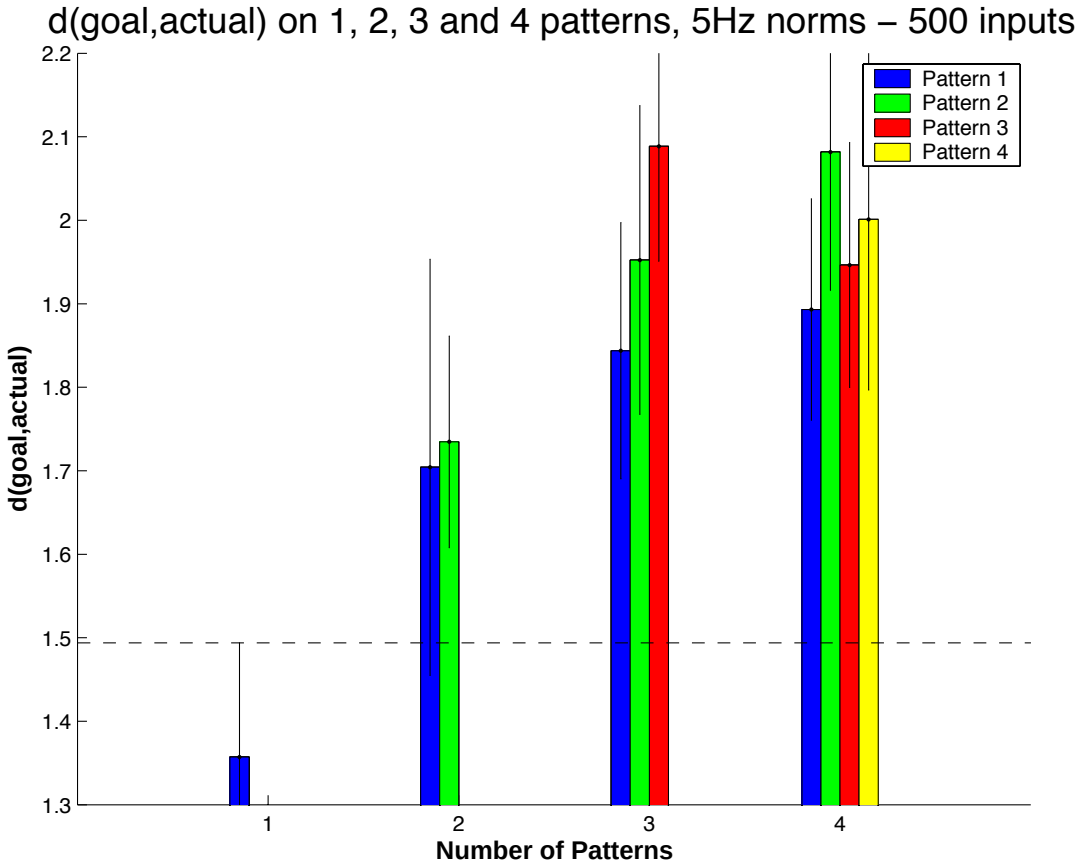


Figure 6-10: d(goal,actual) for 500 input neurons trained on 2 3 and 4 I/O associations — each average over 20 iterations

taneously, while maintaining an acceptable level of accuracy for each individual pattern stored.

6.13 Extrapolation of number of patterns stored

If one considers the results that are shown in figures 6-14, of the increasing networks sizes and the number of patterns they are able to store successfully, and if one were to define a *learnt* pattern as one that is statistically insignificant from the 500 input benchmark case, then one is able to create a plot of the number

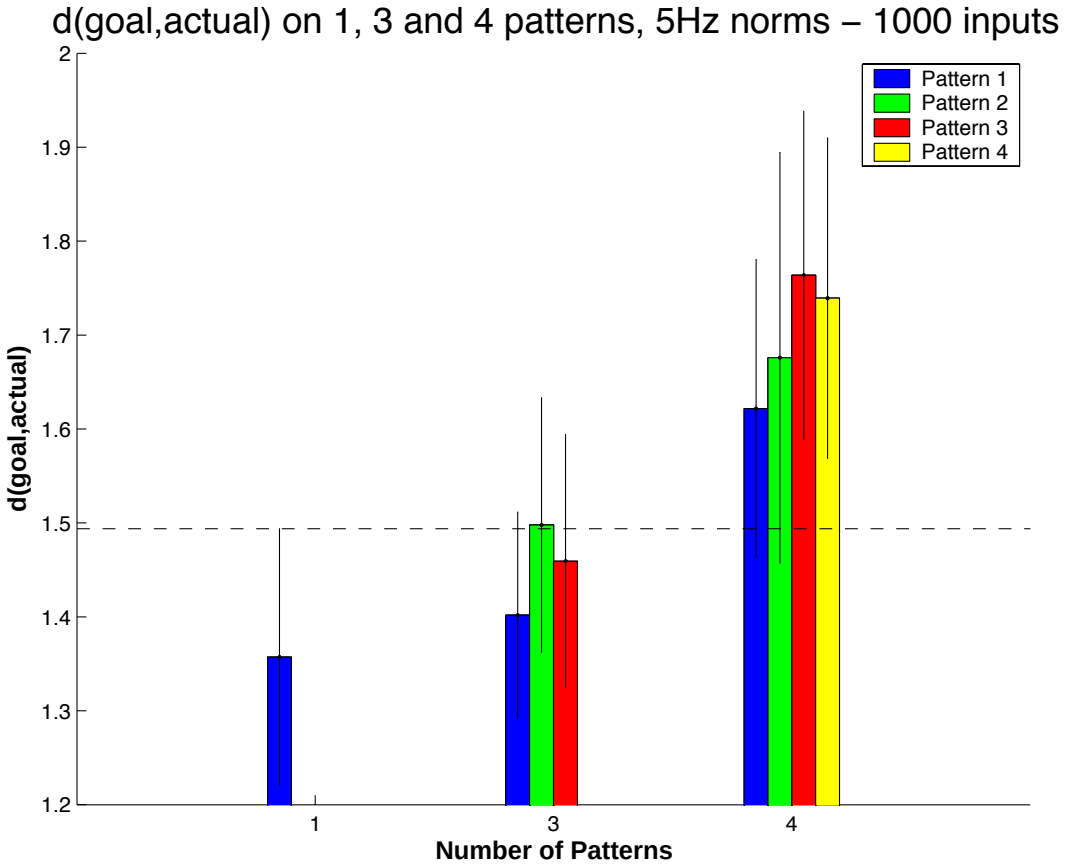


Figure 6-11: d(goal,actual) for 1000 input neurons trained on 3 and 4 I/O associations — each average over 20 iterations

of patterns that can be stored with respect to the number of inputs to the single neuron.

Such a plot can be seen in figure 6-15. It can be seen that the relationship would seem to indicate that in order to store one extra pattern, the number of input to a single neuron must be approximately doubled. The plot contains data from the 500, 1000, 2000 and 4000 input neuron cases. The number of input connections for a single neo-cortical neuron is around $10,000$. If one were to extrapolate these results to $10,000$ synaptic inputs, it can be seen that a single neuron with approximately 10000 input neurons may be capable of storing a maximum number of at least 7 I/O associations, at the $5Hz$ rate.

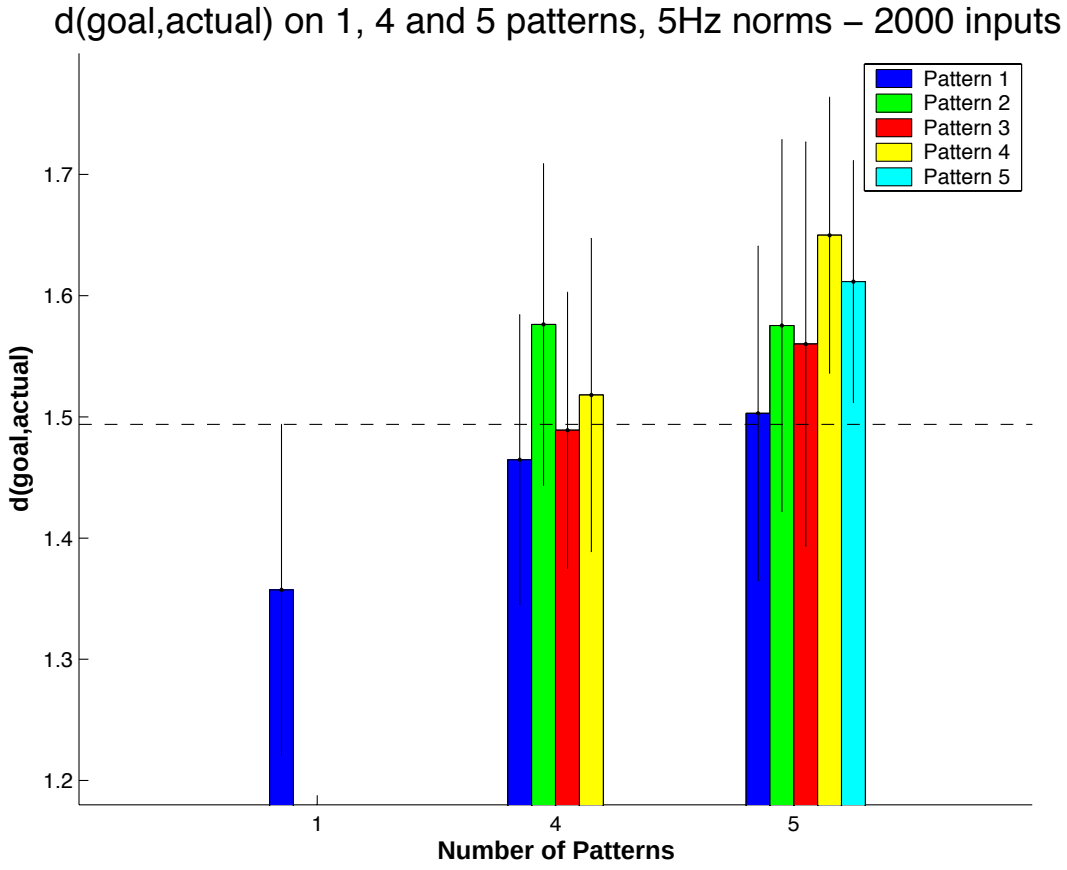


Figure 6-12: d(goal,actual) for 2000 input neurons trained on 3, 4 and 5 I/O associations — each average over 20 iterations

It should be noted that what constitutes a *learned* pattern here is defined as a pattern that is statistically insignificant to the 500 input benchmark case, as previously stated. However, one could equally say that a learned pattern does not need to be this accurate, because due to the possibility of redundancy in large neural networks, it would be possible for many neurons to learn the same output pattern, and so it would not matter so much in this case if the learned pattern was not so accurate because there would likely be other neurons to literally fill in the gaps. One could equally have been less rigorous in defining an accurate pattern, and such an approach would lead to the conclusion that a single neuron of any size could learn more I/O associations than is stated here. The reason for

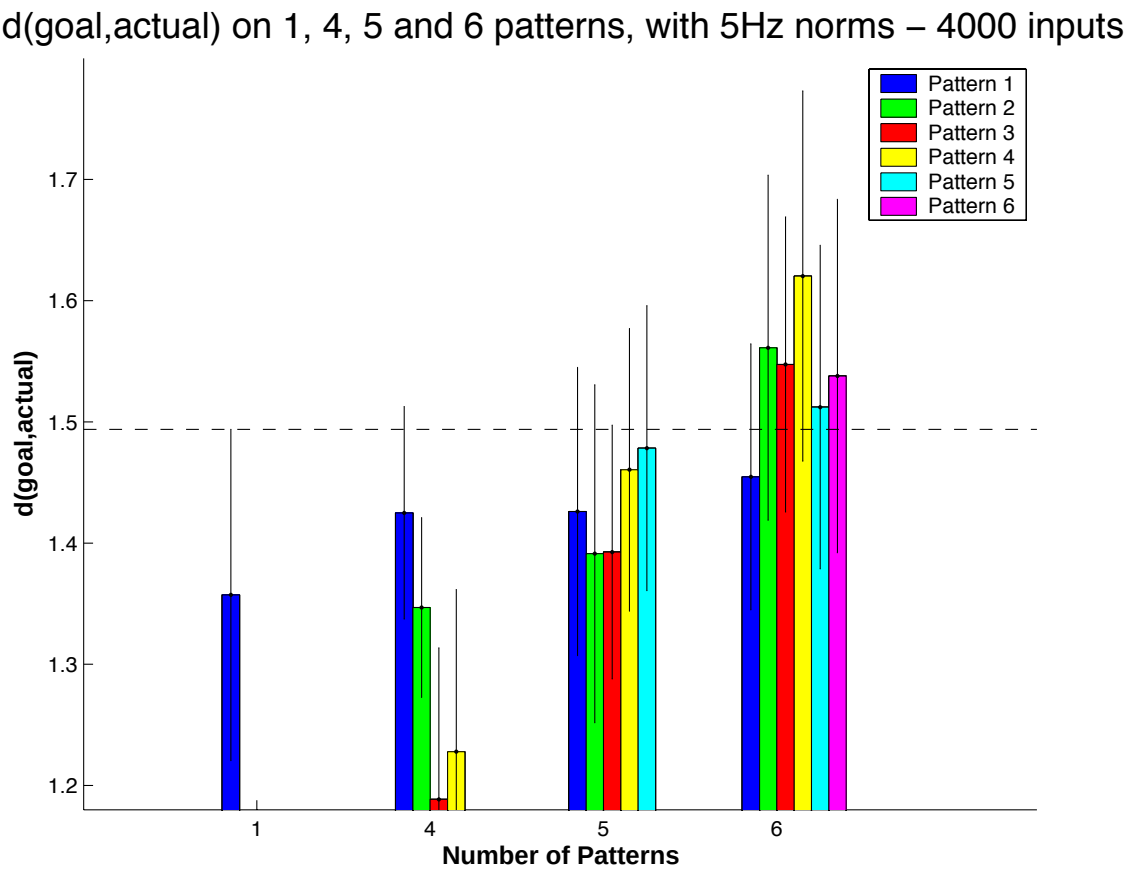


Figure 6-13: d(goal,actual) for 4000 input neurons trained on 4 5 and 6 I/O associations — each average over 20 iterations

defining a learned pattern as it has been here is that it is suitable for the purpose of the work and because a line has to be drawn somewhere.

What can be said for sure from all of these results, is that the 500 input case is capable of learning at least a single pattern simultaneously with a high degree of accuracy. While a single neuron with 1000, 2000 and 4000 inputs can learn a maximum of at least 3, 4 and 6 I/O associations respectively. Also, increasing the number of inputs to a single neuron would appear to improve the accuracy with which the neuron is able to learn a single precise pattern — up to a point, as it can be seen in figure 6-9 that the 4000 input case and the 2000 input case both have a similar average distance result for learning a single pattern. It can

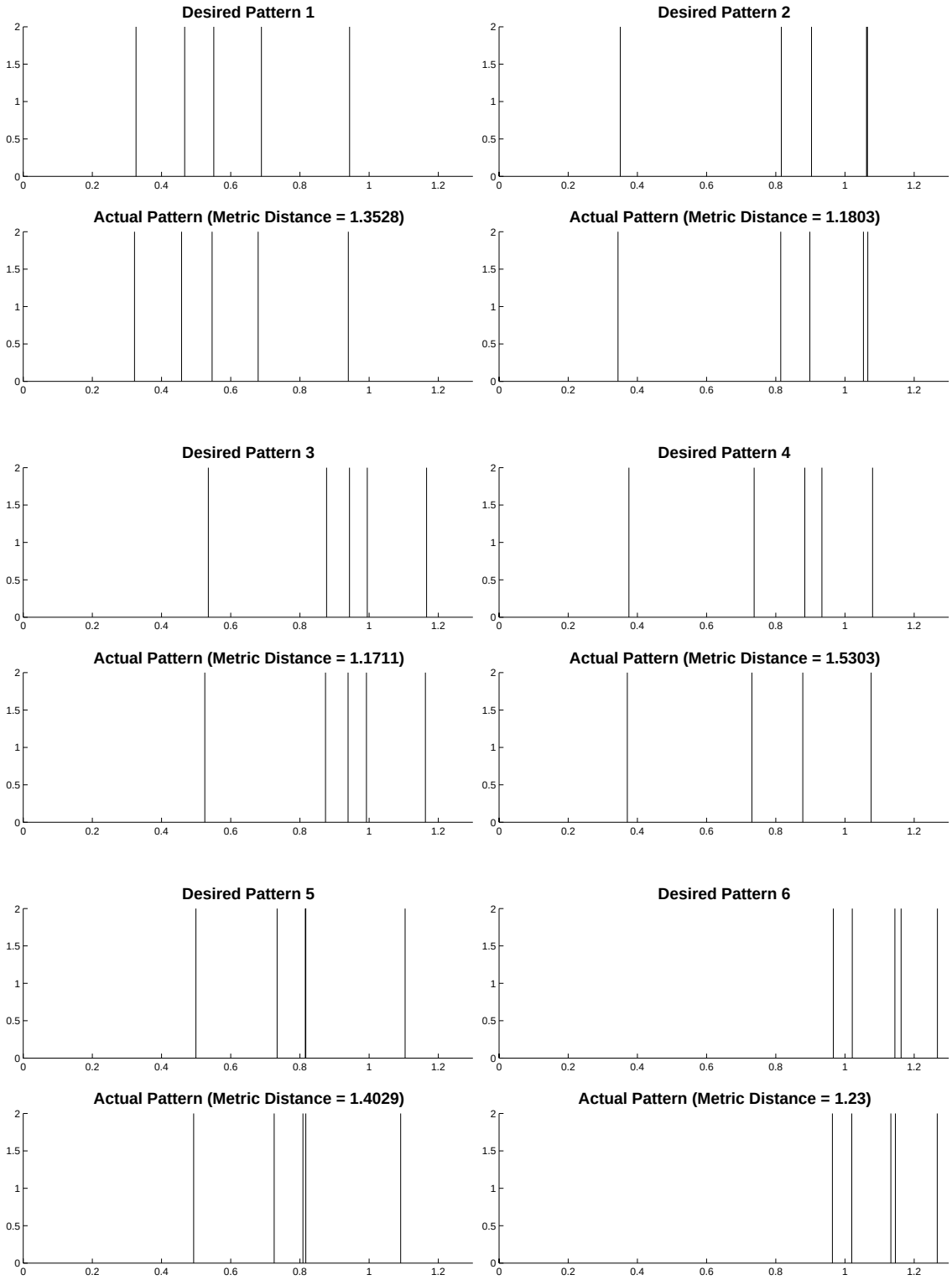


Figure 6-14: 6 raster plots demonstrating the result of using $STDP + N$ learning on the input synapses of a single neuron, with a total of 4000 inputs, in order to simultaneously learn 6 I/O associations.

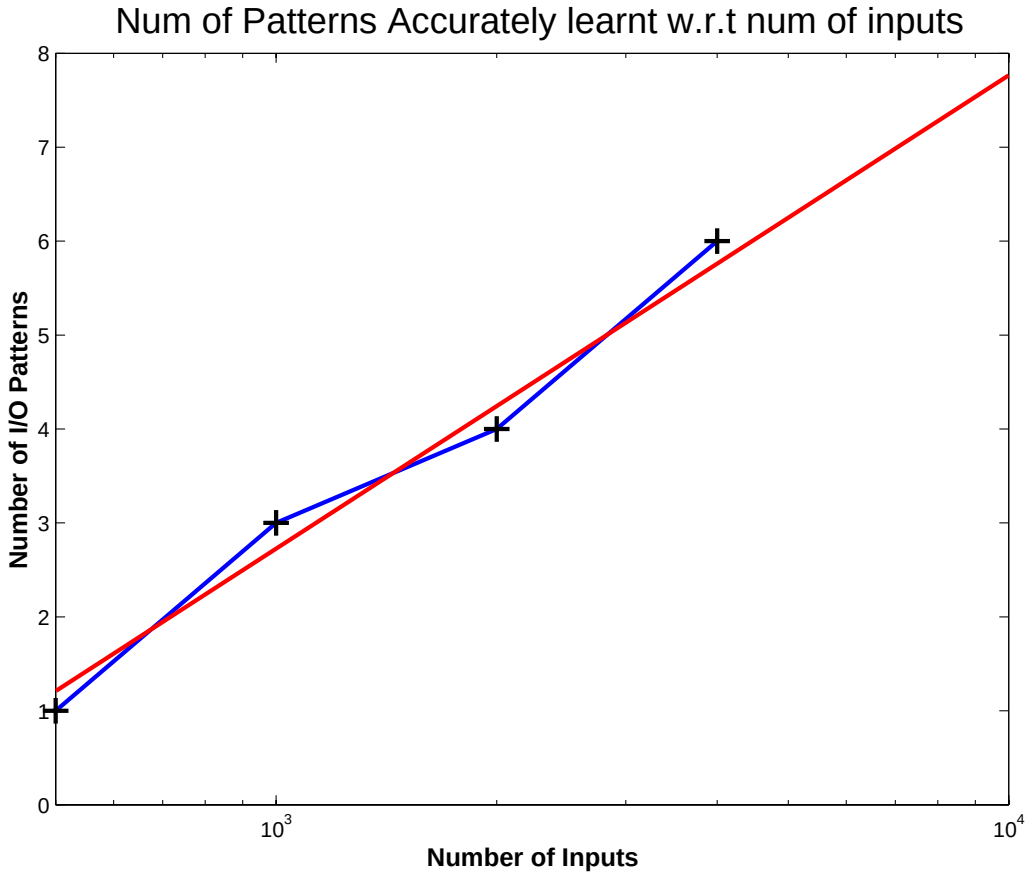


Figure 6-15: This figure shows two things: The blue line is a plot of the number of I/O associations successfully stored, each with an accuracy comparable to the 500 input benchmark case, with respect to the number of input neurons to a single neuron; The red line is an extrapolation from 4000 to 10000 inputs neurons, with the purpose predicting how many of these I/O associations may be learnt by a single neuron in the human neocortex, which typically receives *circa* 10000 input connections.

also be said that the relationship between the number of patterns that a single neuron can learn simultaneously, and the number of input neurons it receives, would appear to be a *log-linear* relationship.

Concerning the choice of the firing frequencies of the input and output spike trains, while the justification for these has already been given previously in this section, 6.8, it should be noted that it is not the concern of the work presented

here to investigate the effects of different frequency choices on the number of patterns which can be successfully learnt by a neuron for any given number of synapses. However, it thought that changing the firing frequencies used here would not alter the findings significantly, but merely skew them in some way. For example, in the extrapolation graph of figure 6-15 it is likely that an increase in the number of spikes in the goal spike train would lead to a decrease in the number of patterns that a single neuron could learn for a given number of inputs. From the results obtained it would be reasonable to say that it is likely that this effect would be constant across the range of the number inputs the neuron receives, and so the whole line would likely be shifted downwards by some amount w.r.t the y-axis.

More research in this area, ideally with access to greater computational power would allow for even larger numbers of inputs to be investigated and also allow for more accurate predictions. More computational power would also allow the learning to be run for a greater number of repetitions, which would also likely serve to reduce the $d(goal, actual)$ values.

6.14 Discussion

In the first part of this chapter, a metric was defined that is based upon the inner product of two spike trains. In the metric section the two problems considered are subproblems of the more difficult problem of constructing given stimulus-response patterns. This can be modelled in the following way. Let S be a time series of spikes on an interval $[a, b]$, and let R be a time series on a subsequent interval $[b, c]$. Suppose that we give S to every neuron in a network. The problem would be to pick the weights to the output in such a way that we get response R at the output of our output neuron. This should be true for a typical internal state of the network. That is to say, in accordance with the ideas of anytime computing, we are not allowed to prepare the network by setting the internal state at the moment when the stimulus S is given. See Natschläger *et al* (2002b), Maass *et al* (2002a).

It was subsequently shown that it is possible to use $STDP + N$ learning on the input synapses of a single neuron to train such a neuron to produce a highly precise spike train output in response to a unique and precise selection of input spike trains. Furthermore it was shown that it is possible for even a single neuron to learn several of these highly precise I/O associations simultaneously, without each of the learnt patterns erasing or interfering in too much of a detrimental manner with each other. Figure 6-14 demonstrated that it is possible in some cases for a single neuron to learn at least 6 I/O associations simultaneously. A neuron is considered to have successfully learnt a collection of I/O associations if the average metric distances $d(goal, actual)$ for all I/O associations have overlapping confidence intervals with the average metric distance of the benchmark case. There also appears to be an increasing difficulty to store multiple patterns simultaneously that can only be partially addressed with increasing the number of inputs to the single neuron. The nature of the log-linear relationship that was discovered, between the number of inputs to the neuron and the number of patterns it can learn simultaneously means that, for every extra I/O association that a neuron is required to learn, there is an exponentially increasing penalty, in terms of the number of inputs a neuron would require to learn all of the I/O associations to a sufficient accuracy. This log-linear relationship was shown in figure 6-15. Figure 6-15 also shows an extrapolation to $10,000$ inputs — a number that reflects the typical number of inputs a neo-cortical neuron may receive (Peters and Jones, 1984).

This work demonstrates the potential computing power and usefulness of single spiking neurons, and that larger networks based upon the inclusion of such well trained individual neurons could provide computationally powerful solutions to realtime computing with spiking neural networks. It also presents a possible limit, based on simulations, on the number of multiple I/O associations that a neocortical neuron can learn.

Chapter 7

Novel application of $STDP + N$ Learning

7.1 Introduction

This chapter examines the effects of, and the possibilities arising from, applying STDP learning to relatively large and highly recurrent networks of LIF neurons. What are the benefits of applying STDP learning? Can it improve the computational power of recurrent neural networks, with respect to using memory capacity as a measure? An improvement in such attributes is desirable, from the point of view that it is important to study the limits of such networks, in order to know what they are capable of, and therefore what applications they may have. The use of an STDP mechanism for learning makes such a study meaningful because of the biological foundation of STDP. The biological foundation of STDP is the reason for selecting it as the learning regime.

The LSM is used as a starting point for the networks considered here, in which learning is applied. The LSM is typically used as a static system from the point of view that, once generated, its parameters such as, synaptic weights, synaptic time delays and connectivity among others, are fixed. Examples of such work can

seen in Maass *et al* (2002a, 2002b, 2004a, 2004b, 2005), Natschläger *et al* (2002a, 2002b) and in Bertschinger and Natschläger (2004).

In general the LSM has been treated as a system to be used for the representation of input stimuli in a more diverse and higher dimensional form so that readout units are able to be trained for salient aspects of the input information stream. An important property, with respect to the ability of an LSM to perform more complex computations, is its *memory capacity*, as defined in Bertschinger and Natschläger (2004). The greater the memory capacity of a LSM, the further into the past it can *remember* — see Bertschinger and Natschläger (2004) for a full and detailed explanation of memory capacity.

What is proposed and demonstrated in this section is a method of applying the $STDP + N$ learning described in chapter 5 in a realistic manner to recurrent neural networks, with the intention of eliciting several functional improvements to the network. Principally, these improvements can be categorised as follows: *i*) An increase in the duration for which the network is able to sustain non-chaotic and specific activity in response to an input spike; *ii*) As a result of *i*), the network can be trained to store, within its spiking activity, the response to multiple individual spikes in a robust manner which maintains the integrity of each spike; *iii*) As a result of *i*) and *ii*), it is shown that the resulting networks may exhibit an enhanced memory capacity over the typical static untrained LSM networks — an enhancement which is shown to increase with the size of the network.

The goal is to train a network of neurons to reproduce a desired goal output of a *Mexican wave* type firing pattern, to which it is exposed during some learning phase. Roughly speaking the recurrent network is divided up into sub-groups of neurons, also referred to here as groups which, during training, are clamped to fire by some external stimulus in a sequential order. The time intervals between the firing of each group may be regular or irregular. Additionally, these intervals may be larger than the rise time of the post synaptic response, in which case, the desired Mexican wave pattern cannot be achieved by direct connections alone.

In the case in which only direct connections are created by the $STDP + N$ learning, incorrect associations/patterns are formed quickly and *clustering* of firing times is observed. In the standard method of STDP application (also used in application of $STDP+N$), the learning is applied to all synapses of all neurons within a network — this is referred to here as $STDP + N_{Type1}$ learning.

Note that, $STDP + N_{Type1}$ is exactly the same learning method as $STDP + N$ learning. The additional type sub-script allows for the definition of different varieties of $STDP + N$ learning. Also defined is a Type 2 learning regime, $STDP + N_{Type2}$, in which the learning locality moves randomly around the network and is restricted to smaller randomly chosen portions of the sub-groups of the network.

It is found that using $STDP + N_{Type2}$ learning, it is possible for the network to learn goal patterns that the traditional $STDP + N_{Type1}$ learning is unable to learn. For example, using $STDP + N_{Type2}$ it is shown that it is possible for groups of neurons to learn to fire outside of the rise-time of the post synaptic response, through the creation of *bridging* neurons, thus avoiding the clustering effect seen using the traditional $STDP + N_{Type1}$ learning. This means that those networks trained with $STDP + N_{Type2}$ learning are able to learn to mimic the goal Mexican wave pattern presented during learning, while $STDP + N_{Type1}$ trained networks are unable to do so, due to the clustering of firing times. It appears that $STDP + N_{Type2}$ learning produces networks capable of sustaining activity for longer periods than those trained with $STDP + N_{Type1}$ learning, and consequently have higher memory capacities.

The resulting $STDP + N_{Type2}$ trained networks can also be used to perform the more complex task of storing a precise spike train, or PST. This involves a precise spike train being presented to the network which, due to it having undergone training on the goal Mexican wave pattern, is able to store each spike within its temporal firing activity and therefore able to preserve the relative temporal position of each spike of the PST.

Furthermore, it is proposed that, provided that the duration for which the net-

work can sustain a single input spike is longer than the duration of the input PST, then a readout neuron or population of neurons can be trained to produce a second distinct PST at a time τ after the last spike of the initial PST. The value of τ is dependent on the size of the network so that, the larger the network, the longer the initial spike train may be stored. Additionally, the readout can be trained using $STDP + N$ as detailed in section 5, such that the second PST is produced *only* in response to the initial PST and to no other input.

The network structures discussed in this chapter could act as the basic building block — a module or node — from which much larger networks could be constructed. A large number of such nodes, with interconnections between nodes could act as a system for the processing of multiple, diverse precise spike trains — which themselves could be thought of as originating from either real world stimulus or from other nodes of the system. Such a system could not only have a huge processing capability for temporal inputs, but would also have the advantage of being trainable on those inputs. New associations between PSTs could be formed 'on-line' — in effect, new memories made. Such structures could also be used to form the basis of timing or counting mechanisms, similar to the Accumulator structure that was built in chapter 4.

7.2 Basic Network Structure and Setup

The recurrent neural networks considered in this section can be described as follows: Consider a fully connected recurrent network of K LIF neurons, where K is the number of LIF neurons in the network. The recurrent network receives inputs from a series of pools of spiking input neurons, $I_1, I_2, ..., I_m$. Each of the spiking input pools of neurons consists of M input neurons, where, in all cases considered here, $M = 1$.

The recurrent network is divided into m *sub-groups*, with each sub-group consisting of k neurons. In addition to receiving input connections from all other recurrent network neurons, the neurons of each sub-group also receive input from

one of the input groups $I_1, I_2, ..., I_m$. Therefore, the characteristic that defines what group a network neuron will belong to is the input neuron from which it receives a connection. The number of neurons in the recurrent network, K, is therefore equal to the product of m and k. A simple rendering of one such setup can be seen in figure 7-1.

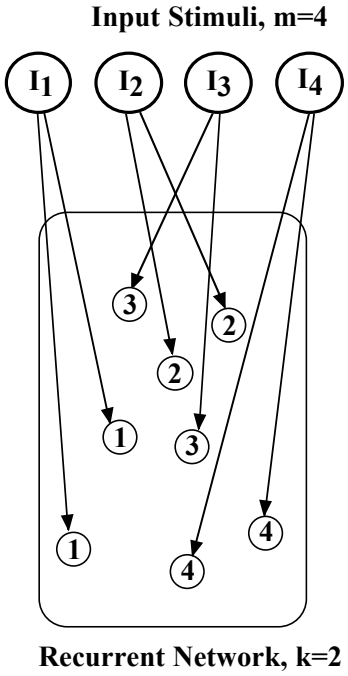


Figure 7-1: A rendering of the experimental setup used for the investigation into $STDP + N$ learning throughout this chapter. The recurrent connections between the neurons of the recurrent network are not shown in this illustration. The number on each of the neurons shown within the recurrent network, represent the sub-group to which the neuron belongs. The sub-group — usually referred to as simply *group* — to which a network neuron belongs is determined by the input neuron from which it receives an input. Network neurons receiving an input from input neuron 1 belong to $group^1$, neurons receiving an input from input neuron 2 belong to $group^2$ and so on. There are as many groups of network neurons as there are input neurons. All groups contain equal numbers of network neurons. In this figure, only 4 groups and 4 inputs are shown, $m = 4$, and there are 2 network neurons per group, $k = 2$.

The above reference to *full* connectivity, means that each neuron in the recurrent

network receives inputs from all other neurons within the recurrent network.

7.2.1 Network Generation

Consider a recurrent network, N, of LIF neurons with full connectivity. The input connection vector to an individual LIF neuron $n \in N$ is defined as $C_n = c_n^1, c_n^2, ..., c_n^{K-1+M}$ where, K is equal to the number of neurons in N, while M is the number of neurons in a single input pool. In the case of the following experiments $M = 1$.

Each LIF neuron in the network N is an excitatory neuron — no inhibitory neurons are used. This is done because the parameters given for inhibitory neurons within CSIM, based upon study of biological inhibitory neurons (Thomson *et al*, 2002), produce inhibitory neurons that respond more quickly to stimuli than excitatory neurons. This faster reaction time means that an inhibitory neuron receiving a stimulus from one of the input neurons, $I_1, I_2, ..., I_m$, will have its synaptic weight modified by an amount that is larger than any modification value for an excitatory neuron in the same situation — see the form of the excitatory segment of the learning window in chapter 5. Therefore, with repeated iterations, the inhibitory neuron dominates and consequently has the effect of dampening down any activity that may have otherwise occurred.

This may suggest that inhibitory neurons have a specialised role within biological neural networks and that they probably should not be treated with the same learning rules as excitatory neurons as is done in Pfister *et al* (2006), and Legenstein *et al* (2005).

7.2.2 Why full Connectivity?

The use of full connectivity of the recurrent neurons is due, in part, to the small size of the sub-groups used throughout this chapter. These groups were discussed and introduced previously in section 7.2 and are also explained in figure 7-1.

This small group size, combined with the fact that the neurons for each sub-group are selected randomly from the whole recurrent network at the initial network generation means that, in order to be sure that a neuron within the network receives enough connections from the relevant sub-group, full, or near full connectivity must be used.

The need for full connectivity can also be thought of as somewhat realistic, by virtue of the fact that within biological neural networks, local connectivity between the same number of neurons considered here would typically be very high. If one considers that a single biological neuron could receive *circa* $10,000$ synaptic connections, then one could arrive at the conclusion that enabling full connectivity in a network of only 200 neurons would be a reasonable implementation.

$STDP + N$ learning discussed in section 5 is applied initially to every single synapse within the recurrent network. This means that, with each neuron receiving hundreds of connections there will be tens of thousands of synapses — in the largest networks considered — to be modified with each iteration of the stimuli through the network and the resulting firing activity.

Computationally, this is quite an intensive task for the resources immediately available. As a consequence the majority of experiments detailed here use networks consisting of only 200 recurrent neurons — for the purposes of obtaining multiple runs in order to demonstrate some form of statistical significance and consistency.

7.3 Choosing Parameters

In order for the $STDP + N$ learning to work successfully when applied to the synaptic weights of a recurrent spiking neural net, it is important that several network and learning rule parameters are correctly tuned. Each of the relevant parameters are outlined and discussed below along with reasoning to support the end choice.

7.3.1 Learning Parameters

Learning parameters are directly associated with the learning function itself. They are: *learning rate*; *excitatory function width*; and *inhibitory function width*. A full list of learning parameters can be found in section A.

Learning rate ϕ was explained in section 5.2.1, it controls the strength of the weight updates. In this chapter it is set to 6.5×10^{-10}, this is ten times greater than the learning rate used in chapter 6. In that chapter the readout neuron was required to learn to replicate a precise spike train consisting of multiple spikes. The success of this task was dependent on a having a learning rate that was large enough to effect substantial changes on the weight vector of the neuron, but also be as small as possible so as to increase the precision of the resulting spike train. Here however, each neuron in the recurrent network is required to learn only a single precise spike and so a larger learning rate can be used.

Excitatory and inhibitory function width together define the total width of the learning window. The total learning window width used is $200ms$, with the width of the inhibitory side of the function being $100ms$ and the width of the excitatory side also being $100ms$, the actual function itself will have decayed to extremely low values before $100ms$ has elapsed.

7.4 Sustaining Network Activity in a Coherent Manner

7.4.1 Experimental Setup

Consider a LSM similar to that shown in figure 7-1. Suppose that each of the neurons within this network are assigned to one of m groups. Neurons are assigned randomly to each group. The defining difference between each group is how it is treated by a training or clamping stimulus. In the case of these experiments, each group receives input *clamping* stimuli from a single input source. The clamping

stimulus is applied in such a way that the groups of neurons fire sequentially — like a Mexican wave of activity in a controlled and consistent manner. So, the group firing sequence would be $group^1, group^2, ...group^m$. The experiments performed here use $m = 10$, and a network size of 50 neurons, giving a group size of $k = 5$ neurons, unless stated otherwise. An illustrative example of this setup can be seen in figure 7-1

It should be noted that some experiments described later in this section will use 20 neurons per group instead of just 5. This four-fold increase means that the learning rate used for $STDP + N$ learning in the 20 neuron per group case is a quarter of the learning rate used in the 5 neuron per group case. The norm of the weight vector to each recurrent network neuron is also reduced by a factor of $\frac{1}{\sqrt{\frac{20}{5}}}$ compared to the 5 neuron per group case. Both of these changes mean that the effect of the synaptic weight changes caused by the $STDP + N$ learning function is relatively the same no matter how many input connections a neuron receives, and that therefore the effect of using more neurons can be isolated and observed.

A biological analogy could be that of a brain region with m afferent stimuli. One could suppose that each of the stimuli originates from other brain regions that are active at different times during some sensory experience, with each stimuli conveying a particular event to the target brain region. In other words, this target region receives a particular multi dimensional input *picture* about an experience.

The firing times of each of the input stimuli are at regular intervals in the following experiments, however one could just as easily substitute these regular intervals with irregular intervals, within an appropriate range, for a more realistic approach. However, for the purpose of demonstrating the concept, only regular intervals are used.

Due to the spatially distributed nature of the neurons that comprise each group and the small group size, it is necessary for connectivity in the network to be high so that it can be guaranteed that neurons belonging to the same group are actually connected. The small network size is used to give a convenient simulation

time, and to ensure connectivity between group neurons, the LSM is connected using full one-to-one connectivity — every neuron receives a single connection from every other neuron in the network. It should be noted that this is not necessarily unrealistic, as biological networks are known to have extremely high connectivity density, and full connectivity in a network of such a small size as is being considered here, is entirely feasible.

The idea is to apply $STDP + N$ learning previously discussed in section 5, to the synaptic connections of the recurrent network while the clamping input firing sequence is applied, with the desired effect that the network learns to recreate this firing sequence when it is presented with only the first group firing. In effect, the network learns to respond as if the entire input stimuli was presented, not just the first group firing i.e. the input sequence becomes imprinted into the network structure.

7.4.2 Implementation of the Clamping Stimulus

As discussed briefly above, in order to ensure that the recurrent network neurons fire at precisely the desired time, they are clamped to fire by some external influence — this is referred to as clamping. The source of the clamping stimuli can be thought of as perhaps originating from external brain regions, as stated in the previous section.

One could actually create these external clamping sources and 'plug' them into the network that is to be trained. However, instead of this, $group^1$ of the network is clamped via an external source, while the remaining groups are made to fire at the desired times by the means of altering their output spike trains in a *hard-coded* manner, i.e. $group^1$ of the network is presented with the clamping stimuli and the remaining groups of neurons all fire sequentially. The effect is the same as if the groups were each being stimulated by their own dedicated input group.

The reason for choosing this approach instead of actually creating and connecting clamping sources is that with the approach used it can be guaranteed that

during training, the activity patterns that the network is exposed to are correct. Whereas, if one were to use the actual clamping sources, one must not only set the connectivity parameters used to connect the clamping sources to the network such that the synaptic weights are of sufficiently large magnitude to cause spiking, but one must also ensure that the network neurons do not experience multiple firings from being exposed to such a large stimulus. In short, one must use some additional mechanism to ensure that the network neurons fire at the desired times, and at these times only.

Investigating how best to train the network effectively using such mechanisms is a separate problem from the one being addressed here and so this approach was discarded in favour of the approach outlined above. For the sake of clarifying the terminology used in the following sections, the clamp that is applied to the neurons will be referred to as the *clamping stimulus*, in spite of the fact that in reality it is not actually an applied stimulus. The effect though should be essentially the same.

In the latter experiments of this chapter, not all of the neurons in a group are clamped. Instead, the output of each of these un-clamped neurons is left untouched. As a result, the un-clamped neurons are free to fire according to the actual input activity they receive from their inputs from the rest of the network.

7.5 Sequential Group Firing

Consider a network of the kind described in the section on experimental setup, section 7.4.1, the network comprises of 50 recurrently connected neurons with each of the 10 groups exposed and trained in turn to their own training stimulus. The neurons of the network are each initially given randomly chosen membrane potentials which decay to resting potential value with time. The clamping stimuli is applied to the network after the membrane potentials have returned to resting values. As a result, the time of the first clamping stimuli that is given to $group^1$, is $0.3s$. The subsequent groups two groups of neurons are clamped to fire at $10ms$

intervals, so at $0.31s$ and $0.32s$. Hebbian learning is applied to the synapses of the network for the duration of the clamping stimulus and the network undergoes 500 iterations of stimulus training. At the end of the presentation of the clamping stimulus, and once the weights have been modified for the current iteration, the network activity is reset before the commencement of the next training iteration. Resetting the network activity means that the membrane potentials of all neurons, and indeed all synaptic activity is reset to initial values, i.e. the membrane potentials are randomly chosen and are allowed to decay to resting levels at the start of the next iteration.

The goal is to train the network so that $group^2$ fires $10ms$ after $group^1$ and $group^3$ fires $10ms$ after $group^2$ *etc.* However, the rise time of the membrane potential of a neuron in the LSM in response to a post-synaptic spike originating from one of its pre-synaptic neurons is $5ms$. This means that the neurons in $group^n$ do learn to fire after, and in response to, $group^{n-1}$ however, $group^n$ will tend to fire within the rise time of $group^{n-1}$ which is $5ms$ and not the required $10ms$. The figure 7-2 shows the response of a single network after such a training scheme. It can be seen in this figure that the first three groups of neurons fire in the correct sequence. While subsequent group all end up firing in one large cluster. The right hand panel in figure 7-2 shows a cumulative rasterised firing plot over 10 networks of the same type seen in the left hand panel which only shows the output of a single network.

7.6 Extending the maximum inter-group interval

In light of what has just been described, a pertinent question would be: Is it possible to not only extend this time interval between successively firing groups of neurons, such that intervals longer than the membrane potential rise time can be achieved, but also to eliminate the *clustering* of firing times so that all groups fire at the correct time and the clamping stimulus is mimicked correctly, while

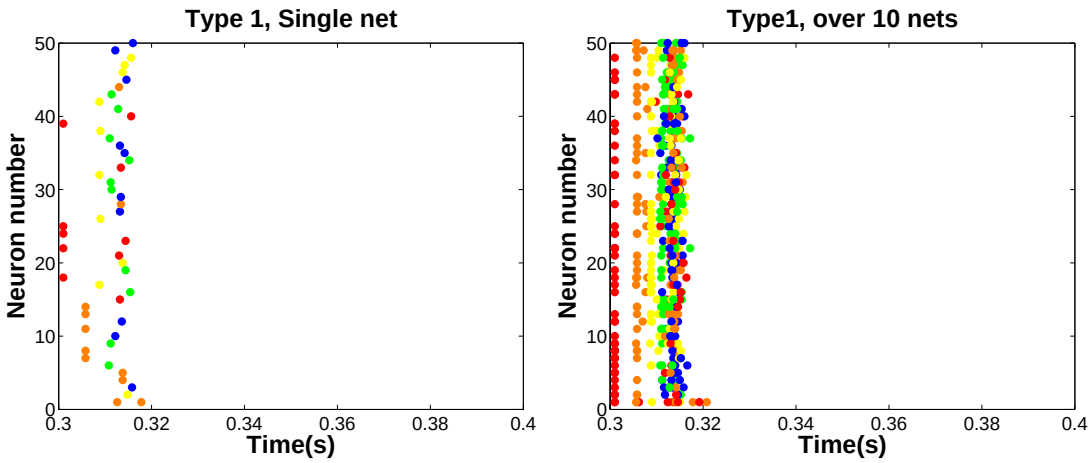


Figure 7-2: Plots of the sequential firing times of ten trained groups of neurons. Groups are colour coded to facilitate the differentiation of the firing times of neurons belonging to the same group from the firing times of other neurons that fire around the same time. A total of five different colors are used, in the order: red; orange; yellow; green; blue. $group^1$ neurons are colour coded red, $group^2$ neurons are colour coded orange, and so on. In networks that consist of more than five groups, the cycle of colours simply repeats itself. Therefore, in networks that consist of more than five groups, some or all colors will be used for more than one group. However, this is not a problem as the function of the colour is merely to distinguish groups that fire close together from each other. The groups were trained to fire with an interval of $10ms$ between them. However, it can be seen that the resulting firing times of the groups are much less than $10ms$ apart. The end result is that the majority of groups fire in a cluster, rather than at their desired sequential times. The groups are unable to learn to fire outside of the rise time, $5ms$, of the membrane potential of the previously firing group of neurons. The left panel shows the rasterised firing times for a single trained network, while the right panel shows the *cumulative* rasterised firing times over 10 networks. $k = 5$ and $m = 10$.

maintaining a realistic and plausible approach?

In order to determine if this can in fact be done, the effect of modifying the clamping technique used and the manner in which the Hebbian learning is applied, will be investigated in the following experiments.

In the previously discussed method, the clamping stimulus was applied to all neurons in a group and this meant that the neurons could only spike at the time the clamp stimulus is active. In this revised approach, a group of neurons has only a portion of its neurons clamped during an iteration of the presentation of the stimulus. 20% of the neurons in the group being trained are chosen randomly at the beginning of each iteration.

As a justification for this method, consider that in reality the neurons of a biological network would likely be receiving an amount of background activity/noise from a number of their numerous input connections. This input noise varies over time. The LIF neurons modeled here have no input noise and clamping a randomly chosen 20% of the neurons in a group is functionally equivalent to the case for a biological network in which all neurons of a group are presented with the input stimulus and in which, the input noise to each neuron is of such a magnitude that upon presentation of the stimulus, the combination of the stimulus and the input noise is such that only 20% of the neurons in the group fire.

This also has the implication that the remaining, un-clamped neurons of the group are, essentially, free to spike according to the prevailing network activity that they receive as they are not being clamped. This is one of the two major modifications whose effect upon the maximum inter-group interval between successively firing groups shall be investigated.

Part of the explanation as to why this may extend the inter-group interval is that it might be leaving some neurons of a group free to spike of their own accord which means that the un-clamped neurons could act as focal points for the clamped neurons of the group. In turn, the Hebbian learning would cause the strengthening of the synapses that relay connections from the un-clamped neurons that are spiking from network activity, to the clamped neurons.

The second modification is made to the method of application of the Hebbian learning and concerns the choice of neuron to which the Hebbian learning is applied. More precisely, whereas previously, the Hebbian learning was applied to all synaptic connections within a network, now the Hebbian learning is only applied to the synaptic connections of selected neurons — specifically, with this modification the Hebbian learning is only applied to the synaptic input connections of the neurons within a group that are active due to the clamping stimulus during the current iteration.

This modification too can be viewed as being realistic, if one considers that it may be the case that Hebbian learning does not occur with the same strength or influence in all parts of a neural network, under all conditions. For example, it could be the case that the strength or indeed, the presence, of Hebbian learning that is applied to a neuron may be dependent on the strength of the input to that neuron. By strength it is meant that a training stimulus would be considered to be a very strong input to a neuron, whereas other input activity is considered to be less strong. The mechanism for this could be explained as follows: assume that an important training stimulus is likely to be represented by more neurons and in a more synchronised manner than, for example, background noise stimulus or less important neuronal signals that may stimulate a neuron. Continuing from this assumption, an important training stimulus would deliver a higher, more temporally focussed current — in the form of synaptic activity — to a target neuron, than less important stimuli. It is proposed that large input currents such as these might actually be *required* to activate a Hebbian learning mechanism at a post-synaptic neuron i.e Hebbian learning may require the increased *energy* of a concerted training stimulus in order to be activated and that in the absence of such a stimulus, Hebbian learning is either inactive at a neuron, or operates at a much lower intensity producing much lower synaptic modifications.

So, in the proposed scenario, a strong training influence — like the clamp stimulus used here — could itself be the initiating event that triggers the synaptic modification mechanism. Without the presence of the strong influence of such a stimulus it could be the case that the synaptic update mechanism either does not

activate or perhaps, operates at a much lower intensity — a dual phase Hebbian learning scenario.

This modified Hebbian learning shall be referred to as $STDP + N_{Type2}$ Hebbian learning, and the previous version shall be referred to as $STDP + N_{Type1}$ Hebbian learning. So to recap: Under $STDP + N_{Type1}$ Hebbian learning, every synapse in the recurrent network is modified by the $STDP + N$ learning function for each iteration of the clamping stimulus/learning scenario; whereas under $STDP + N_{Type2}$ Hebbian learning, the $STDP + N$ learning functions are only applied to the pre-synaptic synapses of those neurons that are clamped during each iteration. It is $STDP + N_{Type2}$ Hebbian learning that enables the neurons of a group to fire at a desired time that is beyond the rise time of the membrane potential of the neurons that comprise the previous group.

In order to complete this analysis, and to investigate all combinations of application the Hebbian learning and the clamping stimulus, two further cases will be considered: i) $STDP + N_{Type3}$ learning, in which Hebbian learning is applied to all synapses in a group and 20% of group neurons are clamped; ii) $STDP + N_{Type4}$ learning, in which Hebbian learning is applied to 20% of the neurons in a group, and the clamping stimulus is applied to all neurons in a group.

In each case the network is presented with the clamping stimulus for 500 iterations — 50 iterations for each of the 10 groups to be trained. Figures 7-3 and 7-4 show synaptic weight values for the same neuron. The neuron in figure 7-3 underwent $STDP + N_{Type1}$ Hebbian learning, while figure 7-4 shows $STDP + N_{Type2}$ Hebbian learning. It can be seen that exposing a neuron to $STDP + N_{Type1}$ Hebbian learning resulted in the weight vector consisting of many varied values. Whereas, for the $STDP + N_{Type2}$ case the weight vector of the neuron consists of mostly very low values, with a select few synapses having a relatively very high strength. The $STDP + N_{Type2}$ learning appears to have learnt a more specific relationship with its input neurons than the $STDP + N_{Type1}$ learning.

In order to obtain a more complete view of what is going on here, figure 7-5 show the resulting histograms of the frequency of occurrence of synaptic weights

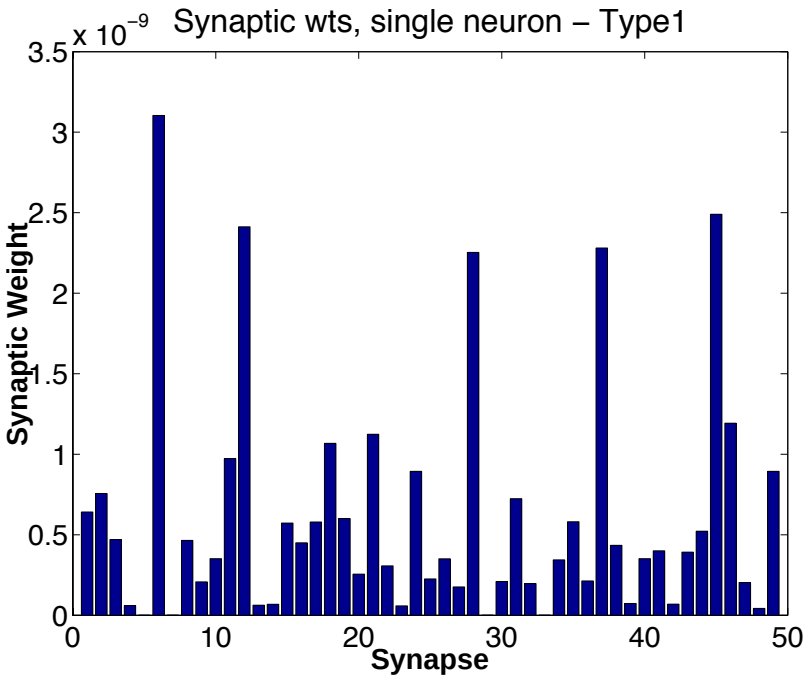


Figure 7-3: Synaptic weight values of the inputs to a $group^3$ neuron for $STDP+N_{Type1}$ Hebbian learning,

throughout the range of their distribution, over 10 networks for $STDP + N_{Type1}$, $STDP + N_{Type2}$, $STDP + N_{Type3}$ and $STDP + N_{Type4}$ learning. There are 1000 bins in each case.

It can be seen from figure 7-5, that $STDP + N_{Type2}$ learning results in a more diverse range of synaptic weight values compared to Types 1, 3 and 4.

7.6.1 Bridging Neurons

Figure 7-6 show the raster plots for the neuron firings produced by the four learning types — each starting from the same initial network. It can be seen that the $STDP + N_{Type2}$ learning has produced a scenario in which the bulk of the neurons in each group fire at $10ms$ intervals, unlike the other learning types. By closer observation of the $STDP + N_{Type2}$ panel in figure 7-6, it can be seen

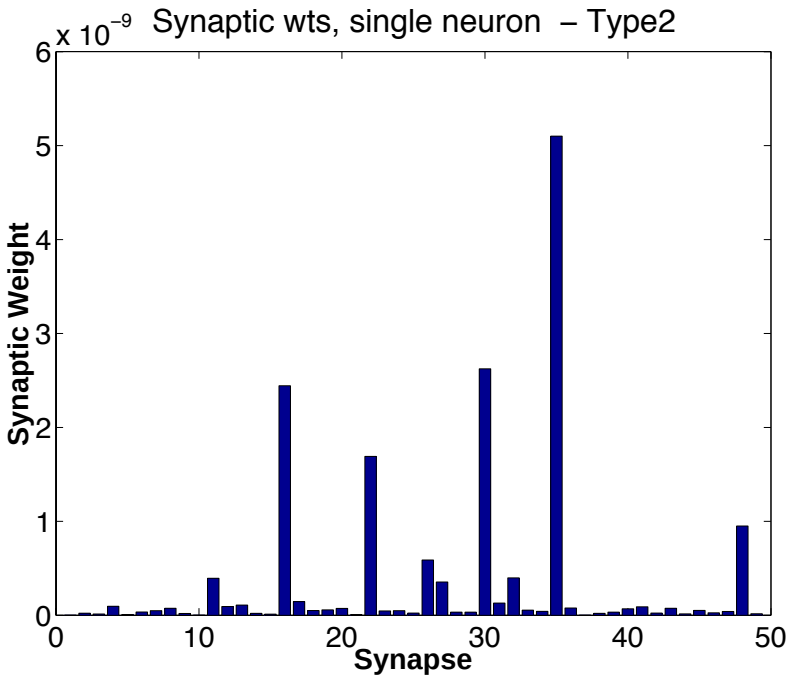


Figure 7-4: Synaptic weight values of the inputs to a $group^3$ neuron for $STDP+N_{Type2}$ Hebbian learning.

that this is accomplished by virtue of the fact that at least one of the neurons of a group learns to spike early in response to the previous group. The remaining neurons have learnt to use this early firing neuron which shall be referred to as a *bridging neuron* as a means to advance the time at which they fire. The bridging neurons fill the gap between the desired firing time of the group and the maximum previously imposed by the rise time.

To reiterate, in the $STDP + N_{Type1}$ case, all neurons in a group are clamped to the desired firing time and Hebbian learning is applied to all synapses of all neurons in the group. Whereas, in the case of $STDP + N_{Type2}$ learning, only a randomly chosen 20% of the neurons in a group are clamped to the desired firing time for any one training epoch, while Hebbian learning is only applied to these clamped neurons.

This covers all of the permutations of the applying the clamping stimulus and

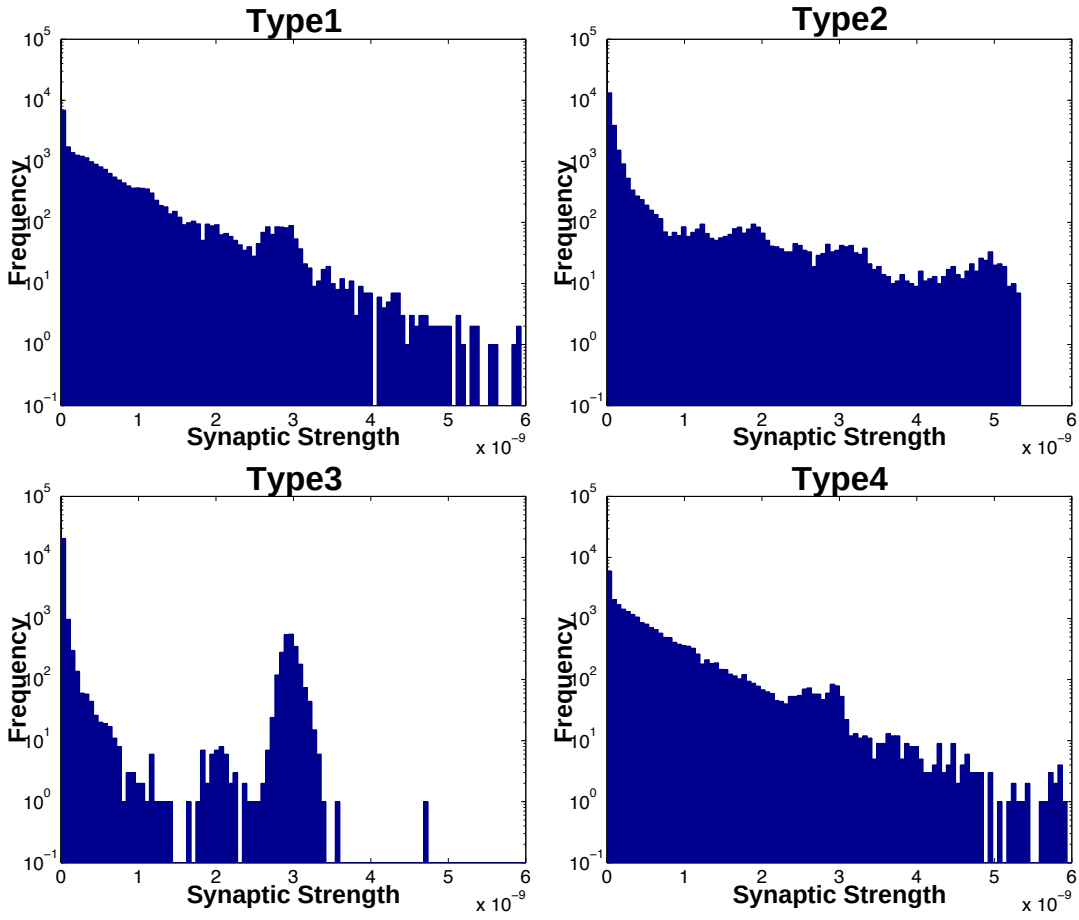


Figure 7-5: Histograms of synaptic weight values produced by Type 1, 2, 3 and 4 learning methods, with values taken over 10 networks in each case. $STDP + N_{Type1}$ and $STDP + N_{Type4}$ learning can be seen to produce virtually identical histogram plots. $STDP + N_{Type3}$ learning produces a distribution in which most weights belong to one of two major populations. $STDP + N_{Type2}$ learning produces synaptic weight vectors in which the majority of the weights are very low, and in which there are a greater number of higher strength weights than in the other learning types. $STDP + N_{Type2}$ learning appears to allow individual post-synaptic neurons to learn more specific relationships with pre-synaptic neurons. Group size for all learning types is 5, while the time interval between each group firing is $0.1s$.

the Hebbian learning. Figure 7-6 shows the raster plot for the case of applying the $STDP + N_{Type3}$ style of learning, as well as the raster plot of spiking activity for the case of $STDP + N_{Type4}$ learning.

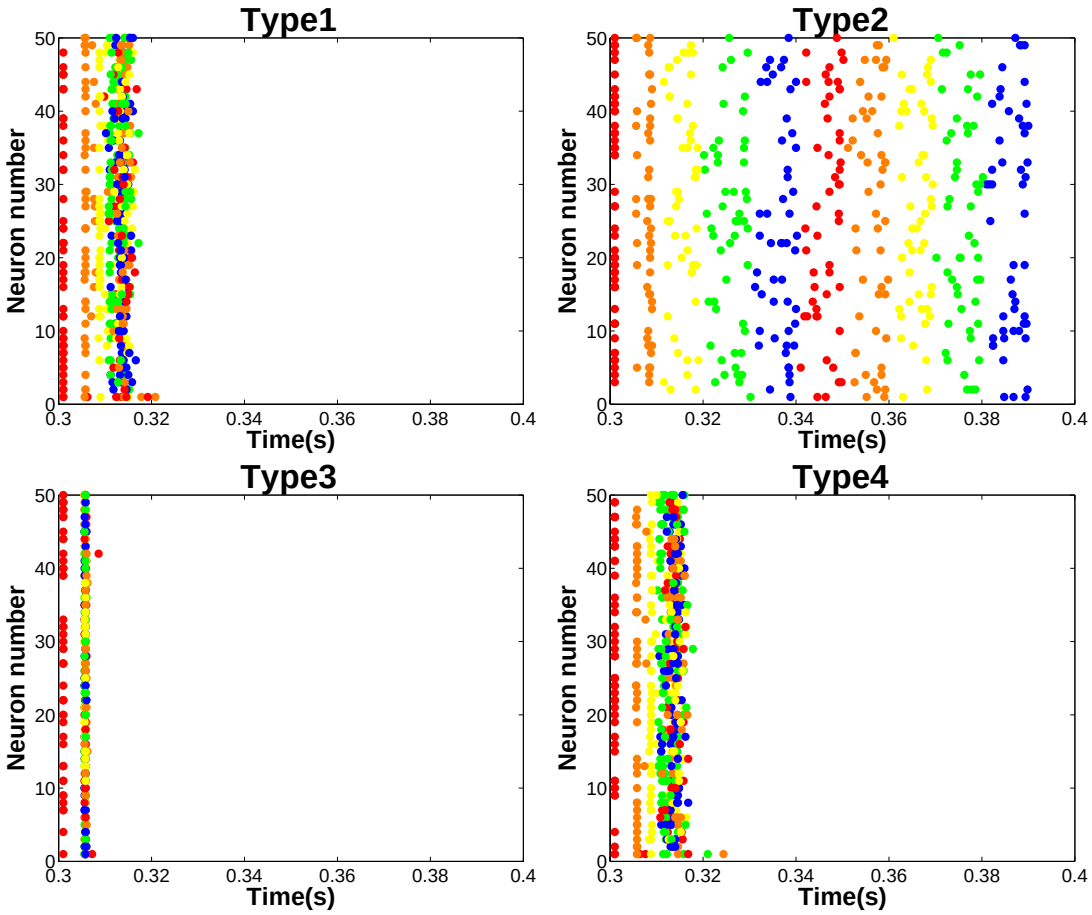


Figure 7-6: Four cumulative firing times plots, each over 10 networks, for fully connected 50 neuron networks with 5 neurons per group with each group trained using $STDP + N$ Hebbian learning to fire at intervals of $0.01s$. $STDP + N_{Type1}$ learning is shown in the top left panel, $STDP + N_{Type2}$ in the top right panel, $STDP + N_{Type3}$ in the bottom left panel and $STDP + N_{Type4}$ in the bottom right panel.

It can be seen that, along with $STDP + N_{Type1}$ learning, both $STDP + N_{Type3}$ and $STDP + N_{Type4}$ learning do not produce the desired result of a group time interval of $0.01s$. Out of the four types of learning just detailed, only $STDP +$

N_{Type2} learning achieves the desired result. Bridging neurons are formed in the $STDP + N_{Type2}$ learning scenario *only*.

Consider the first two firing groups for $STDP + N_{Type1}$, $STDP + N_{Type3}$ and $STDP + N_{Type4}$ learning. It can be seen that the first group learns to fire in response to the input stimuli at $0.3s$ but, the second group is not able to fire at the required time of $0.31s$. In the $STDP + N_{Type2}$ learning case, the majority of neurons in a group become 'tuned' to the smaller number of bridging neurons which enables them to fire at the desired time. However, in the other three learning cases, the group neurons only become tuned to respond to the neurons of the previous group. This severely limits the interval that can be learnt between successively firing groups — only intervals that lie within the rise time of the neuron membrane potential may be learnt.

The clustering of neuron firing times that can be seen in the $STDP + N_{Type1}$, 3 and 4 cases can be explained by considering the width of the excitatory part of the learning window, and noting that the group firings are so close together. This means that the later firing groups become tuned to respond to several preceding groups and because the groups fire in close proximity, the synaptic change caused by each will also be very close and also relatively high. With such a high number of active inputs, extended intervals cannot be learnt.

The upper right panel in figure 7-6 shows the plots of neuron firing times for ten groups of neurons that should fire at $10ms$ intervals, and are colour coded red, orange, yellow, green, blue, red, ..., blue, respectively. The plot is done for 10 randomly generated networks. It can clearly be seen that for each group there exists a *cloud* of neuron firing times for each group of neurons. The density of this cloud is greatest near the desired group firing time. With other neurons being observed firing significantly in advance of this desired time — the bridging neurons.

If one were to increase the inter-group interval between successively firing groups of neurons from $0.01s$ to $0.015s$ for the same network, then it can be seen in the lower left panel of figure 7-7 that the gap is too large to be learnt adequately

and the groups do not learn to fire. A possible explanation for this could be that the number of neurons in each group — in this case 5 — is too small to allow for the learning of larger gaps. It may be the case that increasing the number of neurons per group will increase the length of the maximum inter-group interval which can be learnt.

Consider a similar experimental setup to the one described immediately above. The difference being that the experiment is now performed with 20 neurons per group instead of just 5. For each of the group sizes 10 networks were generated and trained as in the 5 neuron per group case, with inter-group intervals of $0.01s$, $0.015s$ and the spiking output of the network was recorded. It can be seen in the upper right panel of figure 7-7 that, the networks with the larger number of 20 neurons per group are able to learn the group firing interval of $0.01s$ with a far greater number of neurons being able to firing at the desired time and not simply act as bridging neurons. In figure the lower right panel of 7-7, it can be seen that the network with 20 neurons per group is also able to learn the larger group firing interval of $0.015s$, which the networks with only 5 neurons per group were unable to learn.

The networks with 20 neurons per group, *are* able to learn the longer interval group firing times. The number of bridging neurons between two successive groups $group^{n-1}$ and $group^{n}$, that are required for these networks to learn the longer intervals is greater than the entire number of neurons per group of the smaller networks. It would therefore appear that the more neurons that inhabit a group, the greater the number of neurons that can become bridging neurons is, and the greater the number of neurons that are able to fire at the desired time.

However, it should be noted that the maximum length of an inter-group interval is limited by the form of the $STDP + N$ learning window itself. Consider the form of the learning window, first described in section 5. The length of the excitatory portion of the window has decayed appreciably by $20ms$. Therefore, the implication of this for a post-synaptic neuron which fires at a time t, is that it will become increasingly difficult to learn group firing intervals that approach $20ms$ in duration.

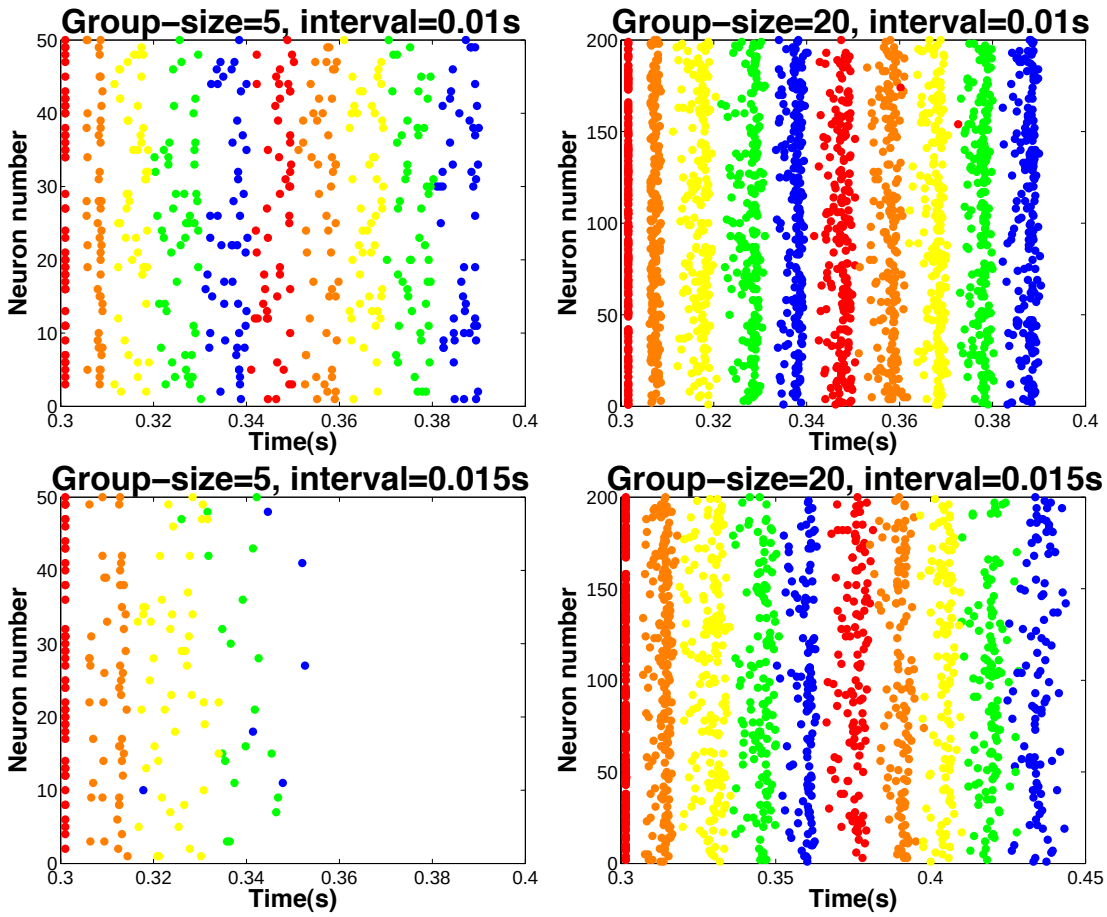


Figure 7-7: The upper panels show the cumulative rasterised firing times over 10 networks, consisting of 10 sub-groups, and 5 neurons per group in the top left panel, and 20 neurons per group in the top right panel, in which the groups have been trained to fire at $0.01s$ intervals. Both the 5 and 20 neuron group sizes are able to learn this interval. The lower panels shows the cumulative rasterised firing times, also over 10 networks with the same setup as in the upper panels but in which the groups have been trained to fire at $0.015s$ intervals. The 5 neuron group size cannot learn the longer interval, while the 20 neuron group size can.

These result appear to show that the dual phase Hebbian learning approach allows for groups of neurons to be trained to respond to past spiking activity on time-scales that are greater than the time-constant of the dynamics of the neuron membrane.

7.7 Extending Duration of Activity

It has just been shown in section 7.6 how, through the use of the application of STDP to the synapses of a recurrent neural network with the setup shown in figure 7-1, it is possible to enhance the response activity of the network neurons so that specific groups of neurons spike at a desired time and in a desired sequence.

With a goal in mind of extending the capabilities of the system and in order for this property to potentially become of more practical use, it will be necessary to create a method that will allow for a system to be trained that enables further extended periods of this firing activity that has already been shown. Figure 7-8 shows a raster plot of the firing times of the neurons in a 250 neuron, fully connected recurrent network, in which each group consists of 5 neurons. The network has been trained as detailed previously in this chapter, and is similar in setup to the network shown in figure 7-1

Figure 7-8 shows that through the use of the co-ordinated wave of network activity, it is possible to prolong the life-span of a single spike injected into $group^1$ of the network for an amount of time that depends on the network size and the inter spike intervals that can be learned — which is itself extended by use of the $STDP + N_{Type2}$ Hebbian learning in favour of the other learning types previously discussed.

7.7.1 Storage of Multiple Spikes

The network just described and whose output was shown in figure 7-8, can be thought of as a basic building block for much larger spike storage systems. For

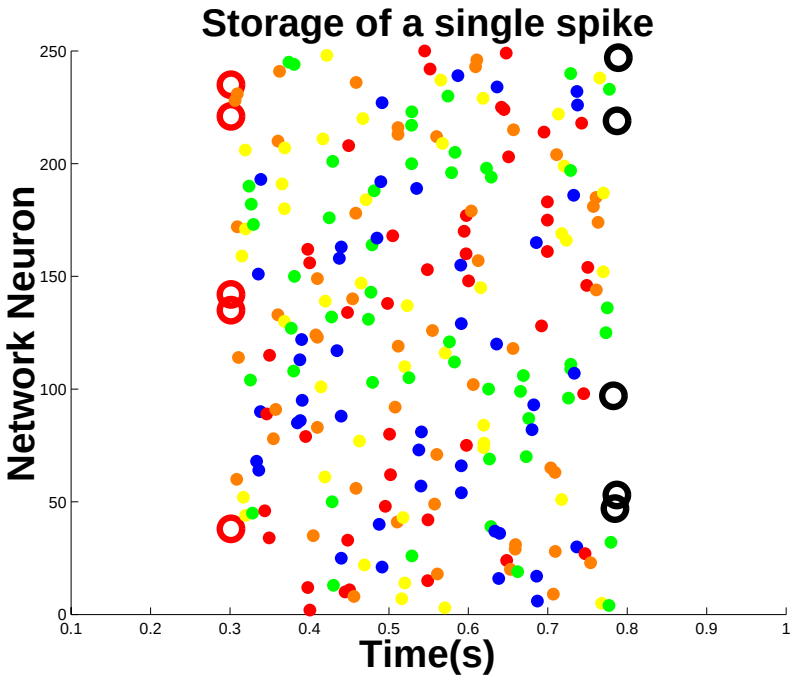


Figure 7-8: Spiking output of a 250 neuron network. A spike is injected into the $group^1$ neurons of the network, it can be seen that the $STDP + N_{Type2}$ trained network is able to sustain the input spike within its network dynamics for an extended period of time. The large red circles in the figure represent the firing times of each of the $group^1$ neurons, while the large black circles represent the firing times of the $group^{50}$ neurons. The single injected spike traverses through all groups of the network successfully. $k = 5$ and $m = 50$.

example, one could connect together a series of such networks in such a way that spiking activity flows through one network and into another and so on, as shown in figure 7-9.

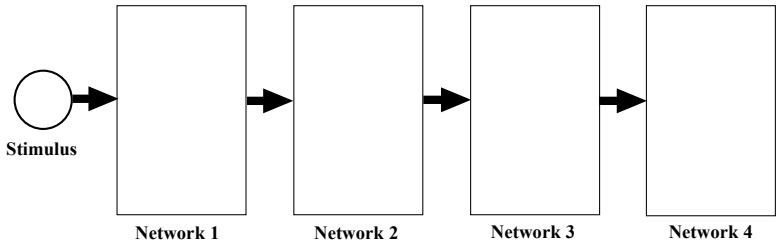


Figure 7-9: A rendering of a multi-network spike train storage system. Several of the networks shown in 7-8 are connected together, to create a system capable of storing entire precise spike trains.

Figure 7-10 shows a raster plot of the spiking network activity for just such a network. The increase in the number of networks from 1 to 4 means that instead of just being able to store single spikes within the dynamics, now, entire precise spike trains my be injected into $group^1$. The plot in figure 7-10 demonstrates that the network is able to respond in a successful and consistent manner to each of the precisely timed spikes of the spike train. Note that the network is capable of holding the spike train within its dynamics for a period of time after the input to the network has ceased.

This simple experiment demonstrates the ability of a correctly trained recurrent network to store a precise spike train within its dynamics and while maintaining the integrity of the spike train.

7.8 Comparison with Static LSM

In Bertschinger & Natschläger (2004), it is shown that the memory capacity of a LSM consisting of 1000 neurons, has a memory capacity equal to between 4 and $5bits$. The 1000 neuron, 200 group network shown in figure 7-10 is able to retain an individual, unique input spike within its network dynamics for a period of 2

seconds. The frequency of the input is $4Hz$, so if one were to assume a time-bin width of $250ms$ for the 3-bit parity task, previously described in chapter 2.6.1, then the network has a maximum memory capacity of 6-bits. However, due to the nature of the network, there is no fading memory. Rather, there is an immediate loss of memory after a spike has been within the network for 2 seconds. So, by their nature, these networks are more *rigid* constructions than the LSM, in that they rely on a high degree of precision in the timings of the hundreds, if not thousands, of neuronal and synaptic firings that constitute their dynamics. The advantage though is that for the full extent of those 2 seconds the spike is completely preserved within the network dynamics. Such networks are limited only by their size — the more neurons they possess, the longer a spike, or indeed a precise spike train may be preserved within their dynamics.

It should be noted that while these *mexican wave* networks appear to have higher memory capacities than a typical basic LSM, and can also be trained to behave predictably, untrained LSM's exhibit more diverse spiking activity, which may itself be useful property.

7.9 Discussion

It was shown that the method of application with which Hebbian learning along with the clamping stimulus are applied, can have a large impact on what a group of neurons may learn about the stimulus. It was shown in figure 7-6 that it was only possible for the sub-groups of neurons of the network to learn to mimic the stimulus if only a selection of neurons within each group actually perform the stimulus during any single training epoch, while the remaining neurons are left free to fire — $STDP + N_{Type2}$ learning. This could be an indication that an appropriate level of noise within the inputs of the network neurons may be considered to be a good thing, as explained in section 7.6. Additionally, and contrary to the accepted methodology of applying Hebbian learning to all synapses of a network at every training epoch, it would appear that the network can only successfully mimic the stimulus input if the Hebbian learning is only applied to

those synapses of the neurons that are activated primarily due to the stimulus. Using a combination of these methods of application of the Hebbian learning and the clamping stimulus figure 7-6 shows that learning is successful only in this case — referred to as $STDP + N_{Type2}$ learning. $STDP + N_{Type2}$ learning is enables groups of neurons to be trained to fire *outside* of the rise time of the membrane potential of the LIF neurons in response to a post-synaptic spike.

It was proposed in section 7.6 that a possible explanation for this could be that Hebbian learning at a synapse could only be initiated by a sufficiently strong input signal and that even if the post-synaptic neuron spikes in response to a weaker input signal, Hebbian learning may not be initiated as the weaker input is unable to activate the learning mechanism. This is a novel proposal for Hebbian learning implementation, and appears to enable learning capabilities that normal Hebbian learning — $STDP + N_{Type1}$ — cannot attain.

In section 7-6, it was shown that $STDP+N_{Type2}$ learning allows for the formation of bridging neurons in each sub-group of neurons. These bridging neurons spike early — before the desired firing time of the group — and enable subsequent firing neurons to strengthen their synaptic inputs to allow them to focus on the bridging neurons. A string of these bridging neurons forms which then allows the majority of neurons in the sub-group to fire at the desired time. The $STDP+N_{Type2}$ panel in figure 7-6 shows the early firing bridging neurons and the higher density of firings around the actual desired group firing time. It was also shown that using a larger value of neurons per sub-group means that more neurons can become bridging neurons and that as a consequence, even greater intervals between sub-group firings can be achieved, than with a smaller number of neurons per group. This can be seen in figure 7-7. Ultimately the maximum length of these intervals is limited by the form of the STDP learning function.

Using a 50 group network, with 5 neurons per group, the trained output of which is shown in figure 7-8, it was shown that, using $STDP + N_{Type2}$ learning, a spike can be stored within the spiking dynamics of the network for an extended period of time, when compared $STDP + N_{Type1}$, $STDP + N_{Type3}$ and $STDP + N_{Type4}$ leaning and also to an untrained, randomly generated network consisting

of the same number of neurons. Expanding on this idea, four such networks were connected together to allow for the storage of a precise spike train consisting of 4 spikes. This expanded network is able to store a spike within its dynamics for a duration of 2 seconds. This extended capacity means that it is possible for a spike train of a realistic duration of $1s$ to be input into the network, stored within the spiking dynamics and used by a readout neuron — or group of readout neurons — for further computations during an interval that does not overlap with the time interval during which the input is active. Larger networks can hold the spike trains for longer periods of time after the input spike train has ceased.

The type of trained network shown in figure 7-10, by virtue of its ability to successfully preserve a number of distinct and precise input spikes within its dynamics, could also be considered as a form of the Accumulator structure that was introduced in chapter 4. In its current form the network would not store spikes indefinitely, as the Accumulator did in chapter 4, rather, it would store them only for a finite period of time before they are lost. However, one could consider the case in which the final firing group of such a network could be linked in some way to the first group, thus providing a structure in which precisely timed spikes are able to cycle indefinitely, unless the network were to be inhibited by some external source. Such structures could act as reliable timing or counting mechanisms within even larger spiking neural networks, and provide these larger networks with regulating mechanisms by which further complex neural activity could be co-ordinated.

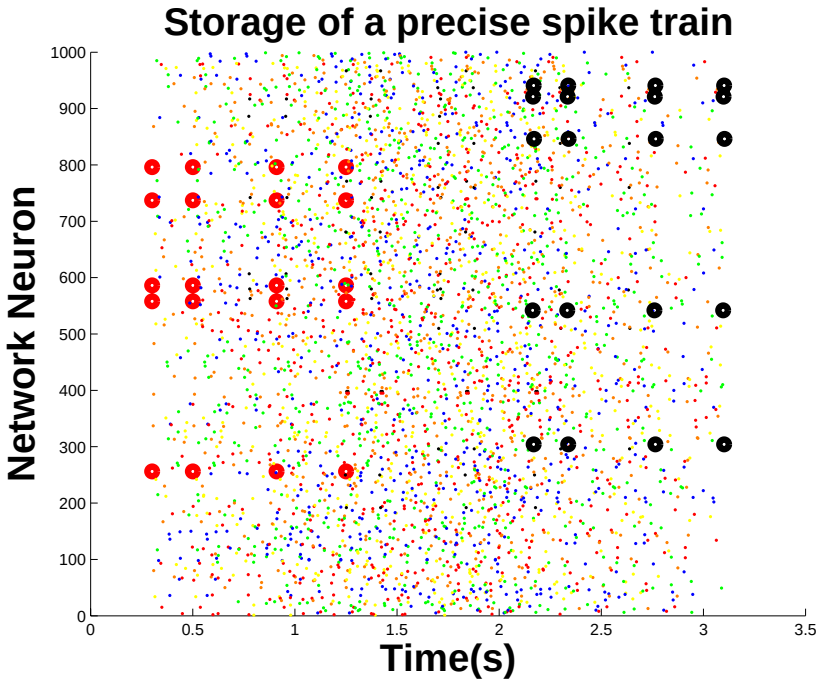


Figure 7-10: Rasterised spiking output of four of the trained, 250 neuron networks previously seen in figure 7-8. The four networks are connected in series so that the firing activity flows though each network and into the next. In the figure the larger red circles represent the firing times of the $group^1$ neurons, while the larger black circles represent the firing times of the last group of neurons, $group^{200}$. The precise firing pattern injected into $group^1$ is replicated, to a large extent, by the precise firing pattern of the neurons of $group^{200}$. This demonstrates that the precise spike train injected into $group^1$ of the network has traversed all groups of the network successfully and that the temporal precision has been preserved. There is an extended period of time, after the last firing of the $group^1$ neurons and before the first firing of the $group^{200}$ neurons, during which, the entire spike train is held within the dynamics of the four connected networks. $k = 5$ and $m = 200$.

Chapter 8

Conclusions and Further Work

The original work in this dissertation has been primarily concerned with investigating what single neurons and recurrently connected networks of neurons can learn and how well they can learn it. Chapter 4 demonstrated a novel application for purpose built networks of spiking Leaky Integrate and Fire neurons, while the majority of the work, chapters 5, 6 and 7, focussed on using a biologically inspired learning regime to investigate learning within networks of spiking Leaky Integrate and Fire neurons.

Chapter 2 introduced the concept of the Liquid State Machine along with significant and relevant examples of the current and recent research in this field. Chapter 2 also discussed some of the aspects of the biological basis for the neuron and network models that are used in this dissertation. The focus of the second background chapter, chapter 3, is a detailed review of recent research into the learning of precise spike time firings using biologically inspired mechanisms.

In this chapter, the results and conjectures of chapters 4, 5, 6 and 7 shall be reiterated and, where appropriate, possible future avenues of research are given.

8.1 Phase Arithmetic Conclusions

Chapter 4, contains original work describing and demonstrating, through the use of experimental simulations, how it is possible to perform simple calculations such as addition and subtraction in neural networks. Moreover, a novel method for running parallel programming constructs within networks of neurons is described.

The motivation for the work in chapter 4 was to apply spiking neurons to a task very different to their typical application, with a view to demonstrating a novel approach to computing using these structures. Section 4.4 defines a very simple parallel programming language. In sections 4.8 and 4.9, the relatively simple structures created out of spiking neurons are outlined, such as: oscillators, switches, synchronisers, accumulators and coincidence detectors. In section 4.10 these basic structures are used to form the basis of the IF, WHILE, SEQ, PAR and ALT constructors. In section 4.11, the basic language previously defined is used to describe the processes of addition, subtraction and multiplication, that could be achieved using differing combinations of instances of the previously defined neural structures.

The ability to create such constructs out of simple neuron models, and have them operate with a high degree of integrity raises the possibility of a novel computing technique. The next step of this work is already being taken in work by Alain Nogaret at the University of Bath (Nogaret, 2004).The work detailed in Nogaret *et al* (2004), involves the use of custom designed and manufactured semiconductor p-n microwires, to create physical structures that can approximate the performance and behaviour of the LIF neurons described in this thesis. With the creation of a workable neuron in silicon, the next step of this work could be to build custom designed arrays of these neurons to create the structures and constructs described in here.

8.2 Convergence and Habituation Conclusions

The reason for undertaking the work in chapter 5 was the need to define a biologically inspired learning regime for use in networks of LIF neurons and to establish the integrity of this learning regime in addition to investigating some of its fundamental aspects from both experimental and mathematical techniques.

The framework for $STDP + N$ learning is introduced in section 5.2 as a biologically inspired learning regime and is implemented with recurrently connected networks of spiking neurons. Integrity of $STDP + N$ learning is established with the evidence presented in section 5.5.2 that showed approximate convergence for $STDP + N$ learning. This convergence is taken as an indication of learning and that information contained within the input regime has become stored within the synaptic weight vectors of the network neurons.

In order to examine the learning from a mathematical point of view the activity link vector L was defined in section 5.3, to define a relationship between the spiking input activity received by a neuron in a recurrent network, and the input weight vector of the neuron. It was shown that when exposed to the input stimulus for a sufficient number of iterations, the weight vector approaches an approximate fixed point/stable state.

It was shown that continued presentation of the input stimulus to the network, combined with the Hebbian $STDP + N$ learning at the synaptic level, and with an appropriate learning rate that, the activity link L becomes more aligned with the weight vector W for each neuron in the recurrent network, according to the alignment measure specified in chapter 5.

Further experimental work in section 5.5.3 and 5.6, showed that for recurrent networks in which the average alignment of L and W over all neurons, was relatively high, then Hebbian and Anti-Hebbian could be considered to be *approximately* equivalent processes. However, in those networks that have low alignment between L and W, Hebbian and Anti-Hebbian cannot be considered approximately equivalent. Section 5.7 demonstrates that networks with $STDP + N$ learning are

sensitive to changes in its input stimulus and that once weights have reached an approximately converged state, it is possible to elicit further significant synaptic modifications by altering the temporal structure of the input stimulus.

8.3 Metric and Multi-Pattern Learning Conclusions

The results of chapter 6 can be divided into two distinct sections. In the first part of the chapter a new metric on spike trains is introduced that is based on the inner product between two spike trains. The motivation for this first section of chapter 6 stems from the need to establish a measure for the effectiveness of the learning in the experiments of the latter part of the chapter. This metric is detailed in section 6.2. It is shown in section 6.4 that, this metric can be successfully used to train a single spiking LIF neuron to produce precise goal spike train in response to a sufficient number of diverse input spike trains. It was shown that, using the iterative process seen in section 6.3.3, the distance between the actual output of the neuron and the goal output — as defined by the metric itself — decreases with the number of repetitions until they are highly similar. Such a metric is continuous over the whole spike train and is a highly effective learning tool, in addition to being a useful continuous measure of the similarity of different spike-trains. In these sections the metric is used to measure the error between the actual output spike train and the goal output spike train, $d(goal, actual)$.

The motivation for the latter part of chapter 6 is borne out by the fact that there has been relatively little research performed that investigates the computational power of an individual spiking neuron, compared to the amount of research that exists on networks of recurrently connected spiking neurons. Therefore, these experiments were performed in an attempt to further the understanding of the computational potential of individual neurons. It was shown in section 6.8 that, the $STDP + N$ learning of chapter 5 can also be used to train a single neuron to

produce a temporally precise spike train, in response to a collection of temporally precise input spike trains. It was shown that this can be achieved by forcing a single neuron to perform a specific goal spike train in response to a selection of input spike trains, the neuron can be trained to produce the precise spike goal spike train.

In section 6.9, the $STDP + N$ learning method was then used to investigate the limit to the number of these precise spike train responses that can be learned *simultaneously* by a single neuron. Again, the metric devised in the earlier part of the chapter is used to indicate how close a given spike train pattern is to the desired goal pattern. The degradation in goal-to-actual spike train similarity as the neuron is forced to learn increasing numbers of I/O associations, increases.It was shown that increasing the number of input spike trains to a single neuron has the result of causing an increase in the number of unique I/O associations that a single neuron can *know* at any time. This is an interesting result, because it demonstrates that even a single neuron can be a powerful computational tool when dealing with many precisely timed stimuli and suggests a massive computational potential if networks of many such neurons, each with many inputs could be trained on temporal patterns in this manner. Extrapolating the results of the work here it is conjectured that it may be possible for a typical neuron of the neocortex with approximately $10,000$ input channels to learn in the region of 7 unique I/O patterns simultaneously. The results obtained appear to show that there exists a log-linear relationship between the number of connections a single neuron receives and the number of I/O associations the neuron can learn without any significant degradation in the recall of each individual I/O association.

8.4 Novel Application of $STDP + N$ Learning Conclusions

In the context of this thesis, chapter 7 ties together concepts and work from each of the previously discussed chapters, to investigate further, the capabilities of

recurrent spiking networks of neurons. In this chapter it was shown that simply applying $STDP + N$ Hebbian learning — now referred to as $STDP + N_{Type1}$ learning — to every synapse in a recurrent neural network, is not necessarily the best, nor perhaps the correct, method of application. This was demonstrated by the experiment in section 7-2.

The motivation for performing the experimental work in this chapter was to investigate the application of Hebbian-type learning from a perspective that has not apparently been previously investigated, with a view to determining if there are any advantages to different methods of application of learning. It was shown in section 7.5 that, using the typical $STDP + N_{Type1}$ learning, it does not appear possible for a group of post-synaptic neurons to be taught to fire outside of the duration of the rise-time of the internal state of a group of pre-synaptic neurons, which makes sense. However, it was shown in section 7.6 that using the $STDP + N_{Type2}$ Hebbian learning method, it does become possible to train groups of neurons to fire outside of the rise-time of the pre-synaptic neurons.

What makes this possible is the formation of bridging neurons in the post-synaptic neuron group. These bridging neurons in each group, allows for a spread in the firing times of neurons within the same group, such that, the bulk of the neurons can fire at the correct time by using the earlier firing bridging neurons as their most influential pre-synaptic neurons. In this way, the $STDP + N_{Type2}$ Hebbian learning allows the neurons to learn more diverse firing patterns. This could prove to be a useful modification to more standard forms of Hebbian learning.

As a consequence of this result it was conjectured that Hebbian learning may in fact, not simply be the result of pre and post-synaptic neural activity, but instead it may also result from the presence of a large training stimulus. It was conjectured that it may be the large magnitude of activity provided to the post-synaptic neuron by such a training stimulus, which acts as the initiating event for Hebbian synaptic modification to occur. It was acknowledged that synaptic modification may occur with pre and post-synaptic activity and without the presence of a large training stimulus but, that the magnitude of any synaptic changes in that scenario may be significantly reduced without the stimulus.

The next finding of this section was that it is possible, using the $STDP + N_{Type2}$ Hebbian technique over many iterations, on many groups of neurons, with each group stimulated in turn by some external stimuli, to create a network able to store an input spike within a mexican wave of network spiking activity, as seen in the experiments and results in section 7-7. The duration for which a spike can be sustained within the network is dependent on the number of neurons in the network, the number of neurons per group and the period of time between the firing each group — note that this period of time is increased by using the $STDP + N_{Type2}$ Hebbian learning in favour of the standard $STDP + N_{Type1}$ learning. Therefore, using $STDP + N_{Type2}$ learning allows fewer neurons to store a single spike for a longer period of time than $STDP + N_{Type1}$ learning.

Furthermore, it was shown that such a trained network can be used to store a precise spike train within a series of waves of network activity, section 7-10. There are a number of possible applications for such a structure, such as: *i*) A form of memory storage for precise spike trains; *ii*) An internal counter, able to be used by other neural structures for either counting or timing purposes; *iii*) An accumulator structure, as first described in chapter 4. Such a structure could act as a fundamental component of many types of neural systems.

The nature of the original work in this dissertation is somewhat disparate in that, while each of the chapters in part II are linked in some way, they also illuminate a unique aspect on the current state of research within spiking neural networks. This dissertation has experimentally demonstrated novel techniques for both improving learning with networks of spiking neurons as well as for using spiking neurons to perform computational tasks in a manner very different to that typically considered. A metric of spike train similarity has been shown that provides a highly useful measure for the similarity of weighted spike trains. This measure has been used successfully by other members of the academic community (Schrauwen and Campenhout, 2007). The dissertation also provides some insight, through experimentation, of the processing potential of single spiking neurons. These experiments indicate the relationship between the potential of a single spiking neuron to learn many temporally precise I/O associations and the number

of synaptic inputs the neuron receives.

8.5 Opinion and Overall Conclusions

I feel that the prognosis for the field of research that encompasses spiking neural networks and precise spike trains is genuinely good. It is a field of research that seems to be growing in terms of popularity with researchers both within academia and commercial research firms. The disposition of these networks to operate on temporally precise data means that there are undoubtedly many possible applications for such technology.

As a researcher in the field of spiking neural networks, I genuinely hope that the field continues to expand and evolve into an even more refined form of science and technology. The potential offered by these networks in terms of improving our understanding of the underlying operational principles of the brain, and even the mind, are hard to ignore. In addition to furthering our understanding, I believe that the field has much to offer in the way of future technologies. For example, in the form of increasingly sophisticated recognition systems, novel and robust creation and storage of temporally complex and precise information within spiking neural networks, improved processing systems for robotics and no doubt many applications that have yet to be considered. It is the opinion of this researcher that the field of spiking neural network research and its related areas have a great deal more to offer in the coming years.

Appendix A

Implementation

The purpose of this chapter is outline and explain the details of the CSIM (IGI Group, 2008) parameters used throughout the simulation work in this thesis.

A.1 CSIM

CSIM (IGI Group, 2008), is a neural network simulation tool written in C++ which is run through a MATLAB interface. It was developed at the University of Graz by members of the IGI group. It is claimed that the software itself is capable of simulating networks containing thousands of neurons and around one million synapses. However, due to the relatively computationally prohibitive requirements of simulating such large scale networks, the typical size of the networks used throughout the work presented here is much smaller. A typical network used for experiments in this thesis may contain a few hundred neurons and $10,000$ synapses.

CSIM has a highly comprehensive range of neuron and synapse types that can be simulated. Some of these models are highly detailed ion-channel based models. Models such as these would typically be used to investigate effects within the neurons or synapses themselves. The work presented in this thesis is focussed on

learning through synaptic modification and, as such, requires the use of simpler neuronal and synaptic models which, can be simulated in recurrently connected group of hundreds of neurons. These models are discussed below but, if further information is required on these or indeed any other model within CSIM, such information may be found in the CSIM reference manual, (IGI Group, 2008).

A.1.1 The Chosen Neuronal Model

The neuron model of choice for this thesis — unless stated otherwise — is the Leaky Integrate and Fire, LIF, model. This model of neuron has an internal state variable measured in Volts that is *charged* by post-synaptic pulses from the input synapses that are incident on the neuron. If the internal state variable crosses a threshold value, the neuron emits a pre-synaptic spike and the internal state undergoes *hyperpolarisation* — this means that the internal state is reset to a value that is below the resting potential. If the neuron receives no further synaptic stimulus, then the internal state gradually rises back the resting potential level over time. The behaviour of the model is governed by the following differential equation which, is taken from Gerstner and Kistler (2002):

$$\tau_m \frac{dV_m}{dt} = -(V_m - V_{resting}) + R_m \cdot (I_{syn}(t) + I_{inject} + I_{noise})$$

in which V_m is the membrane potential of the LIF neuron, $V_{resting}$ is the potential to which the membrane will tend to if given no external stimuli, I_{syn} is the total current contributed by the synapses incident on the neuron, I_{inject} is a background current and I_{noise} is a random gaussian noise with zero mean and a given variance.

At the onset of hyperpolarisation, it is highly unlikely that synaptic stimulus could cause a neuron to fire as the internal state is set to such a low value. The time that the internal state takes to recover to normal resting levels is determined by the time constant. The period of time during which the neuron cannot fire is known as the *absolute refractory period*.

Table A.1 shows the complete set of parameters for the LIF neuron model. In this particular model, upon crossing the threshold the internal state is reset to the potential V_{reset} and is held there for a period of time that is given by $T_{refract}$ — this period of time represents the absolute refractory period, during which it is impossible for the neuron is fire. After the absolute refractory period has passed the internal state falls from V_{reset} back to the resting potential V_{rest} is no other stimuli occurs, this is known as the *relative refractory period*, during which it is possible for a post-synaptic spike to cause the neuron to fire, but it is less likely than if the internal state were at V_{rest}.

LIF neuron model parameters	
Parameter Name	Parameter Value
Capacitance	$3e^{-08}F$ (default value)
Resistance	$1e^{06}\Omega$ (default value)
V_{thresh}	$0.015V$ (default value)
V_{rest}	$0V$ (default value)
V_{init}	$0.013839V$ (default value)
V_{reset}	$-0.0001V$ (default value of $0.0138916V$ in latter expts of chapter 6)
$T_{refract}$	$0.003s$ (default value)
I_{noise}	$0A$
I_{inject}	$1.41e^{-08}A$ (in range $1.35e^{-08}A$ - $1.41e^{-08}A$ in chapter 7)
type	2 for excitatory, 1 for inhibitory
V_m	Membrane voltage (V)
I_{syn}	Total synaptic input current (A)

Table A.1: This table lists the parameters used in CSIM for the LIF neurons used throughout the experimental work presented.

A.1.2 Learning Parameters

Learning parameters			
Parameter Name	Chapter 5	6	Chapter 7
Learning rate ϕ	6.5×10^{-10} [1], $\frac{6.5 \times 10^{-10}}{20}$ [2]	6.5×10^{-11}	6.5×10^{-10}
norm (*Input to Net*)	20×10^{-9}	9×10^{-9}	6×10^{-8} [3]
norm (*Net to Net*)	20×10^{-9}	n/a	6.6×10^{-9} [4]
CScale (*Input to Net*)	4	Inf	Inf
Lambda (*Input to Net*)	10	Inf	Inf
WScale (*Input to Net*)	3	9×10^{-7}	3
CScale (*Net to Net*)	10 [1], 4 [2]	n/a	Inf
Lambda (*Net to Net*)	1.2 [1], 2 [2]	n/a	Inf
WScale (*Net to Net*)	1	n/a	9×10^{-7}
Clamp CScale	n/a	inf	n/a
Clamp lambda	n/a	inf	n/a
Clamp WScale	n/a	9×10^{-7}	n/a
Weight sign fixing used	No	Yes	Yes
% excitatory neurons	100	100	100

Table A.2: This table provides a reasonably comprehensive list of both CSIM and learning regime parameters used in each relevant chapter.

Where *norm* is the value to which the weight vector of individual neurons are set, CScale, C, WScale, Lambda, neuron noise and % excitatory neurons all refer to actual CSIM parameters involved in network generation. For chapter 5 the input neuron is positioned such that it will connect to between 10% and 20% of the network neurons, for any given set of the input connectivity values. All

[1]This is the learning rate for the convergence experiment in section 5.5.2 and the Habituation experiment in section 5.7.

[2]This is the learning rate for the equivalence experiments in chapter 5. It is set to a lower value than the Habituation learning-rate because the learning is to take place over a larger number of iterations and the lower rate means that the weight changes are much finer than with the larger weight.

[3]This is the norm for the *group*[1] neurons (the only neurons that receive an actual input from an external spiking source, se section 7.4.2). Where the group size is 5 neurons. For a group size of 20 this value becomes 30×10^{-9}, see section 7.4.1.

[4]This is the norm used for the network neurons that do not belong to *group*[1], an where the group size is 5 neurons. For a group size of 20 this value becomes 3.3×10^{-9}, see section 7.4.1.

values for learning rates and norm are initial values and alter with network sizes according to the rules described in the relevant chapter.

A.1.3 The Chosen Synaptic Model

As with the neuronal models, there are numerous synaptic models within the CSIM framework. All synapses used throughout this thesis are of the CSIM type *StaticSpikingSynapse*. The attributes of this synapse are as follows: Firstly, the synapse has a weight w that is a measure of the *strength* of the synapse — the larger the weight of a synapse is, then the larger the magnitude, of the post-synaptic spike that is contributed to the internal state of the post-synaptic neuron, will be. The weight of a synapse is randomly generated. The synapse also has a time delay parameter t_{delay} measured in milliseconds typically $2ms$. Longer time delays mean that the synapse will take a longer time before transmitting the post-synaptic spike to the post-synaptic neuron.

CSIM allows for the simulation of networks of LIF neurons that are interconnected by many synapses. Networks can be generated in which any given neuron can be connected to, and receive connections from, many other neurons within the network.

The vast majority of the work presented in this thesis uses the LIF neuronal model with the *staticspikingsynapse* model of CSIM, to which the $STDP + N$ learning described in chapter 5 is applied, using the above parameters unless otherwise stated. Full and complete descriptions of all CSIM models and parameters can be found in the CSIM user manual (IGI Group, 2008).

Appendix B

Publications

The following is a compilation of the published and submitted original work derived from the research contained within this thesis:

- Carnell, A., 2008. How much can one neuron learn? - An investigation of the learning of multiple precise I/O spike train associations. Neural Networks, submitted.
- Carnell, A., 2008. An analysis of the use of Hebbian and Anti-Hebbian Spike Time Dependent Plasticity learning functions within the context of recurrent spiking neural networks. Neurocomputing, accepted.
- Carnell, A. and Richardson, D., 2005. Linear algebra for time series of spikes. Proc. ESANN 2005, Bruges, Belgium, 27-29 April 2005, p.363-368.
- Carnell, A. and Richardson, D., 2007. Parallel computation in spiking neural nets. Theor. Comput. Sci, 386(1-2), p.57-72.

Bibliography

[1] Abeles, M. & Goldstein, M.H., 1977. Multispike train analysis. *proc. IEEE*, 65, p.762-773.

[2] Allman, J.M., Hakeem, A., Erwin, J.M., Nimchinsky, E. & Hof, P., 2001. The anterior cingulate cortex: the evolution of an interface between emotion and cognition. *Annual New York Academy of Sciences*, 935, p.107-17.

[3] Allman, J., Hakeem, A. & Watson, K., 2002. Two Phylogenetic Specializations in the Human Brain. *The Neuroscientist*, 8(4), p.335-346.

[4] Alpha Med Sciences, 1995-2010. *MED 64 Applications*[online]. Available at: `http://www.med64.com/applications.html` [accessed 1 June 2008].

[5] Alpha Med Sciences, 1995-2010. *MED 64 Applications: Synaptic Activity - LTP & CSD*[online]. Available at: `http://www.med64.com/applications/synaptic.html` [accessed 1 June 2008].

[6] Anderson, J., Lampl, I., Reichova, I., Carandini, M. & Ferster, D., 2000. Stimulus dependence of two-state fluctuations of membrane potential in cat visual cortex. *Nat Neurosci*, 3(6), p.617-621.

[7] Auer, P., Burgsteiner, H. & Maass, W., 2002. Reducing communication for distributed learning in neural networks. In Jos 'R. Dorronsoro, ed. *Proc. of the International Conference on Artificial Neural Networks - ICANN 2002,*

volume 2415 of Lecture Notes in Computer Science. Madrid, Spain, 27-31 August, 2002, Springer, 2002, p.123-128.

[8] Bell, C., Han, V., Sugawara, Y. & Grant, K., 1997. Synaptic plasticity in a cerebellum-like structure depends on temporal order. *Nature* 387, p.278-281.

[9] Bertschinger, N. & Natschläger, T., 2004. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7), p.1413-1436.

[10] Bi, G. & Poo, M., 1998. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci*, 18, p.10464-10472.

[11] Bi, G. & Poo, M., 1999. Distributed synaptic modification in neural networks induced by patterned stimulation. *Nature*, 401, p.792-796.

[12] Bi, G., Poo, M., 2001. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annu Rev Neurosci*, 24, p.139-166.

[13] Bienenstock, E.L., Cooper, L.N. & Munro, P.W., 1982. Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *J Neurosci*, 2(1), p.32-48.

[14] Bindal, A. & Hamedi-Hagh, S., 2007. The design of a new spiking neuron using dual work function nanowire transistors. *Nanotechnology* 18, p.1-12.

[15] Binzegger, T., Douglas, R.J. & Martin, K.A., 2004. A quantitative map of the circuit of cat primary visual cortex. *J Neurosci*, 24(39), p.8441-8453.

[16] Bliss, T.V. & Lømo, T., 1973. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *J. Physiol*, 56, p.232-331.

[17] Blum, L., Cucker, F., Shub, A. & Smale, S., 1997. *Complexity and Real Computation*. Secaucus, NJ: Springer-Verlag NewYork, Inc.

[18] Braak, H., Braak, E., 1992. The human entorhinal cortex: normal morphology and lamina-specific pathology in various diseases. *Neuroscience Research, Shannon*, 15, p.6-31.

[19] Brody, C. & Hopfield, K., 2003. Simple networks for spike-timing-based computation, with application to olfactory processing. *Neuron*, 37, p.843-852.

[20] Carnell, A., 2008. An analysis of the use of Hebbian and Anti-Hebbian Spike Time Dependent Plasticity learning functions within the context of recurrent spiking neural networks. *Neurocomputing*, *accepted*.

[21] Carnell, A., 2008. How much can one neuron learn? - An investigation of the learning of multiple precise I/O spike train associations. *Neural Networks*, *submitted*.

[22] Carnell, A. & Richardson, D., 2005. Linear algebra for time series of spikes. *Proc. ESANN 2005*, Bruges, Belgium, 27-29 April 2005, p.363-368

[23] Carnell, A. & Richardson, D., 2007. Parallel computation in spiking neural nets. *Theor. Comput. Sci*, 386(1-2), p.57-72.

[24] Carr, C. E. & Konishi, M., 1990. A circuit for detection of interaural time differences in the brain stem of the barn owl. *J. Neurosci*, 10, p.3227-3246.

[25] Chandra, R. & Optican, L.M., 1997. Detection, classification, and superposition resolution of action potentials in multiunit single-channel recordings by an on-line real-time neural network. *Biomedical Engineering, IEEE Transactions on*, 44, p.403-412.

[26] Cooke, S.F. & Bliss, T.V., 2006. Plasticity in the human central nervous system. *Brain*, 129(7), p.1659-73.

[27] DeFelipe, J., Fariñas, I., 1992. The pyramidal neuron of the cerebral cortex: Morphological and chemical characteristics of the synaptic inputs. *Prog. Neurobiol*, 39, p.563-607.

[28] Delorme, A., Gautrais, J., Van Rullen, R. & Thorpe, S., 1999. SpikeNET: A simulator for modelling large networks of integrate and fire neurons. *Neurocomputing*, 26-27, p.989-996.

[29] Derrida, B., 1987. Dynamical phase transition in non-symmetric spin glasses. *J. Phys. A: Math. Gen.*, 20, p.721-725.

[30] Derrida, B., & Pomeau, Y., 1986. Random networks of automata: A simple annealed approximation. *Europhys. Lett.*, 1, p.45-52.

[31] Derrida, B., & Weisbuch, G., 1986. Evolution of overlaps between configurations in random boolean networks. *J. Physique*, 47, p.1297.

[32] Drá bek, E.F. & Zhou, Q., 2000. Using co-occurrence statistics as an information source for partial parsing of Chinese. *Proceedings of the second workshop on Chinese language processing: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics*, 8 October, Hong Kong.

[33] Duda, R.O., Hart, P.E. & Storck, D.G., 2001. *Pattern classification*. 2nd ed. New York: Wiley.

[34] Eccles J.C., 1984. *Cerebral cortex*. In E.G. Jones, A. Peters, eds, 2, p.1-36. New York: Plenum Press.

[35] Ferster, D., 1987. Origin of orientation-selective EPSPs in simple cells of cat visual cortex. *J Neurosci*, 7(6), p.1780-1791.

[36] Ferster , D., Chung, S. & Wheat, H., 1996. Orientation Selectivity of thalamic input to simple cells of cat visual cortex. *Nature*, 380, p.249-252.

[37] Fransén, E., 2005. A synapse which can switch from inhibitory to excitatory and back. *Neurocomputing*, 65-66, p.39-45.

[38] Froemke, R.C. & Dan, Y., 2002. Spike-timing-dependent synaptic modification induced by natural spike trains. *Nature*, 415, p.433-438.

[39] Gaiarsa, J.L., Caillard, O., & Ben-Ari, Y., 2002. Long-term plasticity at GABAergic and glycinergic synapses: mechanisms and functional significance. *Trends in Neuroscience*, 25, p.564-570.

[40] Gerstner, W. & Kistler, W., 2002. *Spiking Neuron Models Single Neurons, Populations, Plasticity*, Cambridge: Cambridge University Press.

[41] Gorman, R.P. & Sejnowski, T.J., 1988. Analysis of Hidden Layer Units in a Layered Network Trained to Classify Sonar Targets. *Neural Networks*, 1, p.75-89.

[42] Gutig, R., Aharonov, R., Rotter, S. & Sompolinsky, H., 2003. Learning input correlations through non-linear temporally asymmetric Hebbian plasticity. *Journal of Neuroscience*, 23, p.3697-3714.

[43] Haas, J.S., Nowotny, T. & Abarbanel, H.D.I., 2006. Spike-Timing-Dependent Plasticity of Inhibitory Synapses in the Entorhinal Cortex. *J Neurophysiology*, 96, p.3305-3313.

[44] Hæusler, S. & Maass, W., 2007. A statistical analysis of information processing properties of lamina-specific cortical microcircuit models. *Cerebral Cortex*, 17(1), p.149-162.

[45] Haussler, D. & Opper, M., 1995. Mutual information and Bayes methods for learning a distribution. *In Proc. Workshop on the Theory of Neural Networks: The Statistical Mechanics Perspective*, World Scientific.

[46] Haykin, S., 1999. *Neural networks: A comprehensive foundation*. 2nd ed. Upper Saddle River: Prentice Hall.

[47] Hebb, D. O., 1949. *The Organization of Bahaviour*. New York: Wiley.

[48] Herz, A.V.M., Gollish, T., Machens, C.K. & Jaeger, D., 2006. Modelling single-neuron dynamics and computations: A balance of detail and abstraction. *Science*, vol 314.

[49] Hilgetag, C., Burns, G., ONeill, M. & Scannell, J., 2000. Anatomical connectivity defines the organization of clusters of cortical areas in the macaque monkey and the cat. *Philos Trans R Soc Lond B Biol Sci*, 355, p.91-110.

[50] Hinton, G.E., Sejnowski, T.J., 1986. Learning and Relearning in Boltzmann Machines. In J. L. McClelland, D. E. Rumelhart, eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, Cambridge, MA: MIT Press.

[51] Hoare, C.A.R., 1985. *Communicating Sequential Processes*. International, London: Prentice Hall.

[52] Hof, P.R. & Van der Gucht, E., 2006. The Structure of the Cerebral Cortex of the Humpback Whale, Megaptera novaeangliae (Cetacea, Mysticeti, Balaenopteridae), *The Anatomical Record*. 290(1), p.1-31.

[53] Hopfield, J. J., 1995. Pattern recognition computation using action potential timing for stimulus representation. *Nature* 376, p.33-36.

[54] Hubcl, D.H. & Wiesel, T.N., 1968. Receptive fields and functional architecture of monkey striate cortex. *J. Physiology*, 195, p.215-243.

[55] IGI group, 2008. *CSim: A Neural Circuit Simulator, User Manual* [Online]. Available at:
`http://www.lsm.tugraz.at/csim/index.html` [accessed at 1 June 2008].

[56] Inmos Limited, 1998. *Occam2 Reference Manual*. Hemel Hempstead: Prentice-Hall International.

[57] Izhikevich, E.M. & Desai, N.S., 2003. Relating STDP to BCM. *Neural Computation*, 15, p.1511-1523.

[58] Jaeger, H., 2001. The "echo state" approach to analysing and training recurrent neural networks. *German National Research Center for Information Technology*, GMD Report 148.

[59] Jaeger, H., 2002. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. *Technical report, German National Research Center for Information Technology*. GMD Report 159.

[60] Jezzard, P., Matthews, P.M. & Smith, S.M., eds, 2003. *Functional Magnetic Resonance Imaging: An Introduction to Methods*. New York: Oxford University Press.

[61] Johansson, R. & Birznieks, I., 2004. First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature Neuroscience*, 7, p.170-177.

[62] Jones, E.G., 2000. Microcolumns in the cerebral cortex. *Proc Natl Acad Sci USA*. 9 May 2000, 97, p.5019-5021.

[63] Kaas, J.H. & Krubitzer, L.A., 1991. The organization of extrastriate visual cortex. In B. Dreher & S.R. Robinson, eds. *Vision and Visual Dysfunction, Neuroanatomy of the Visual Pathways and Their Development*. Vol 3. London: Macmillan Press, 1991, p.302-323.

[64] Kempter, R., Gerstner, W. & van Hemmen, J.L., 1999. Hebbian learning and spiking neurons. *Phys. Rev. E*, 59(4), p.4498-4514.

[65] Langton, C.G., 1990. Computation at the edge of chaos. *Physica D*, 42, 12-37.

[66] Legenstein, R., N äger, C. & Maass, W., 2005. What can a neuron learn with spike-timing-dependent plasticity? *Neural Computation*, 17(11), p.2337-2382.

[67] Maass, W., 1996. Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, 8(1), p.1-40.

[68] Maass, W., Legenstein, R. & Bertschinger, N., 2005. Methods for estimating the computational power and generalization capability of neural microcir-

cuits. In L. K. Saul, Y. Weiss, and L. Bottou, eds. *Advances in Neural Information Processing Systems*, vol 17, p.865-872. MIT Press,

[69] Maass, W. & Markram, H., 2004. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4), p.593-616.

[70] Maass, W., Natschläger, T. & Markram, H., 2002a. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11), p .2531-2560.

[71] Maass, W., Natschläger, T. & Markram, H., 2002b. A model for real-time computation in generic neural microcircuits. In S. Becker, S. Thrun, and K. Obermayer, eds, *Proc. of NIPS 2002, Advances in Neural Information Processing Systems*. Vancouver, Canada, 9-14 Dec 2002, MIT Press, 2003, p.229-236.

[72] Maass, W., Natschläger, T. & Markram, H., 2004a. Computational models for generic cortical microcircuits. In J. Feng, ed. *Computational Neuroscience: A Comprehensive Approach*, Boca Raton: Chapman & Hall/CRC, p.575-605.

[73] Maass, W., Natschläger, T. & Markram, H., 2004b. Fading memory and kernel properties of generic cortical microcircuit models. *Journal of Physiology*, 98(4-6), p.315-330.

[74] Magerman, D.M. & Marcus, M.P., 1990. Parsing a natural language using mutual information statistics. *Proceedings of the 8th National Conference on Artificial Intelligence*. Boston, Massachusetts, USA, 29 July - 3 August 1990, AAAI Press/MIT Press.

[75] Markram, H., Lübke, J., Frotscher, M. & Sakmann, B., 1997. Regulation of synaptic efficacy by coincidence of postsynaptic AP and EPSP. *Science*, 275, p.213-215.

[76] Mattia, M. & Del Giudice, P., 2000. Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. *Neural Computation*, 12, p.2305-2330.

[77] McKay, M.R., Smith, P.J. & Collings, I.B., 2006. New Properties of Complex Noncentral Quadratic Forms and Bounds on MIMO Mutual Information. *Proc. of the IEEE Int. Symp. on Information Theory (ISIT)*. Seattle, USA, 9-14 July 2006, p.1209-1213.

[78] Mehta, M.R., Lee, A. K. & Wilson, M. A., 2002. Role of experience of oscillations in transforming a rate code into a temporal code. *Nature*, 417, p.741-746.

[79] Moldakarimov, S.B., McClelland, J.L., Ermentrout, G.B., 2006. A homeostatic rule for inhibitory synapses promotes temporal sharpening and cortical reorganization. *Proc Natl Acad Sci USA*, vol 103, no 44, p.16526-16531.

[80] Mountcastle V.B., 1997. The columnar organization of the neocortex. *Brain* 120, p.701-722.

[81] Natschläger, T., Maass, W. & Markram, H., 2002a. The "liquid computer": A novel strategy for real-time computing on time series. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8(1), p.39-43.

[82] Natschläger, T., Markram, H., and Maass, W., 2002b. Computer models and analysis tools for neural microcircuits. In W. Ktter, ed. *Neuroscience Databases: A Practical Guide*. Norwell, MA: Kluwer Academic Publishers, p.121-136.

[83] Neto, J.P., Siegelmann, H. & Costa, J.F., 2003. Symbolic processing in neural networks. *Journal of the Brazilian Computer Society*, 8(3), p. 58-70.

[84] Nimchinsky, E., Gilissen, E., Allman, J.M., Perl, D.P., Erwin, J.M., Hof, P.R., 1999. A neuronal morphologic type unique to humans and great apes. *Proc Nat Acad Sci*. April 1998, 96, p.5268-73.

[85] Nimchinsky, E., Vogt, B.A., Morrison, J. & Hof, P.R., 1995. Spindle neurons of the human anterior cingulate cortex. *J Comp Neurol*, 355, p.27-37.

[86] Nogaret, A., Lambert, N.J., Bending, S.J. & Austin, J., 2004. Artificial Ion Channels and Spike Computation in Modulation Doped Semiconductors. *Europhys.Lett*, 68, p.874.

[87] Ojemann G.A. & Schoenfield-McNeill, J., 1998. Neurons in Human Temporal Cortex Active with Verbal Associative Learning. *Brain and Language*, 64, p.317-327.

[88] Pakkenberg, B. & Gundersen, H.J.G., 1997. Neocortical neuron number in humans: effect of sex and age. *J. Comp. Neurology*, 384, p.312-320.

[89] Pakkenberg, B., Pelvig, D., Marner, L., Bundgaard, M.J., Gundersen, H.J.G., Nyengaard, J.R. & Regeur, L., 2003. Aging and the human neocortex. *Exp. Gerontology*, 38, p.95-99.

[90] Panzeri, S., Peterson, R., Schultz, S., Lebedev, M. & Diamond, M., 2001. The role of spike timing in the coding of stimulus location in rat somatosensory cortex. *Neuron*, 29, p.769-777.

[91] Peacock, M.J.M. & Collings, I.B., 2003. Mutual Information Analysis of Turbo Equalizers for Fixed and Fading Channels. *Proc. of the IEEE Int. Conf. on Communications (ICC), Communication Theory Symposium*. Anchorage, USA, May 2003, p.2938-2942.

[92] Peltarion, 2008. *Synapse*[online]. Available at: `http://www.peltarion.com/products/synapse/?gclid=CK_7mMeZ2ZYCFQO2FQodhWac3g` [accessed 1 Nov 2008].

[93] Peters, A., 1994. Cerebral Cortex, Vol. 10, Primary Visual Cortex of Primates. In A. Peters & K.S. Rockland, eds. New York: Plenum, p.1-36.

[94] Peters, A., Jones, E.G., 1984. Classification of cortical neurons. In A. Peters, E.G. Jones, eds. *Cerebral Cortex*. London: Plenum Press, p.107-121.

[95] Pfister, J.P., Toyoizumi, T. & Barber, D., 2006. Wulfram Gerstner, Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Computation*, 18(6), p.1318-1348.

[96] Ramón y Cajal, S., 1902. Estructura del tuberculo cuadrigemino posterior, cuerpo geniculado interno y vias acusticas centrales. *Rev Tri Microgra f*, 6, p.207-227.

[97] Ramón y Cajal, S., 1955. Histologie du sisteme nerveux de lhomme et des verte bre s, 1909 /1911. *Reprinted by Instituto Ramón y Cajal del C.S.I.C. Madrid*, 1955.

[98] Rieke, F., Warland, D., de R. van Steveninck, R. & Bialek, W., 1997. *Spikes, Exploring the Neural Code*. Massachusetts: MIT Press.

[99] Roberts, W.M. & Hartline, D.K., (1975). Separation of multi-unit nerve impulse trains by a multi-channel linear filter algorithm. *Brain Res.* 94, p.141-149.

[100] Rose, J., 1926. Allocortex bei Tier und Mensch; die sogenannte Riechrinde beim Menschen und beim Affen. *J Psychol Neurol (Leipzig)*, 34, p.261-401.

[101] Rosenblatt, J.F., 1962. *Principles of neurodynamics*. New York: Spartan Books.

[102] Schrauwen, B. & Campenhout, J.V., 2007. Linking non-binned spike train kernels to several existing spike train metrics. *Neurocomputing*, 70, p1247-1253.

[103] Sereno, M.I., Dale, A.M., Reppas, J.B., Kwong, K.K., Belliveau J.W., Brady, T.J., Rosen, B.R. & Tootell, R.B.H., 1995. Borders of multiple visual areas in human revealed by functional magnetic resonance imaging. *Science*, 268, p.889-893.

[104] Seung, S., 2003. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40, p.1063-1073.

[105] Shannon, C.E., 1948. A mathematical theory of communication (parts I and II). *Bell System Technical Journal*, 27, p.379-423 and 623-656, July and Oct.

[106] Siegelmann, H., 1999. *Neural Networks and Analog Computation, Beyond the Turing Limit*, Basel: Birkhauser.

[107] Sigillito, V.G., Wing, S.P., Hutton, L.V. & Baker, K.B., 1989. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10, p.262-266.

[108] Smith, L., 2006. Implementing neural models in silicon. In A. Zomaya, ed. *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*. Springer US, p.433-475.

[109] Song, S. & Abbott, L., 2001. Column and map development and cortical remapping through spike-timing dependent plasticity. *Neuron*, 32, p.339-350.

[110] Song, S., Miller, K. & Abbott, L., 2000. Competitive Hebbian learning through spike-time-dependent synaptic plasticity. *Nature Neuroscience*, 3, p.919-926.

[111] SPSS, 2008. *SPSS Neural Networks*[online]. Available at: `http://www.spss.com/statistics/neural_networks/` [accessed 1 Nov 2008].

[112] Ster, B. & Dobnikar, A., 1996. Neural Networks in Medical Diagnosis: Comparison with Other Methods. In: A. Bulsari, et al., eds, *Proceedings of the International Conference on Engineering Applications of Neural Networks (EANN96)*. London, UK, 17-19 June 1996, p.427-430.

[113] Thomson, A.M., West, D.C., Wang, Y. & Bannister, A.P., 2002. Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2–5 of adult rat and cat neocortex: triple intracellular recordings and biocytin labelling in vitro. *Cereb Cortex*, 12(9), p.936-953.

[114] Toyoizumi, T., Pfister, J.P., Aihara, K. & Gerstner, W., 2007. Optimality Model of Unsupervised Spike-Timing-Dependent Plasticity: Synaptic Memory and Weight Distribution. *Neural Computation*, 19(3), p.639-671.

[115] van Rossum, M.C.W., Bi, G.Q. & Turrigiano, G.G., 2000. Stable Hebbian learning from spike timing-dependent plasticity. *J. Neuroscience*, 20, p.8812-8821.

[116] van Rossum, M.C.W., 2001. A Novel Spike Distance. *Neural Computation*, 13, p.751-783.

[117] Victor, J.D. & Purpura, K.P., 1996. Nature and Precision of Temporal Coding in Visual Cortex: A Metric-Space Analysis. *Journal of Neurophysiology*, 76, p.1310-1326.

[118] Victor, J.D., 2005. Spike train metrics. *Current Opinion in Neurobiology*, 15, p.585-592.

[119] Wang, D.L., 2000. On connectedness, a solution based on oscillatory correlation. *IEEE Transactions on Neural Networks*, 11, p. 935-947.

[120] Widrow, B. & Hoff, M.E., 1960. Adaptive switching circuits. *IRE WESTCON Convention Rec.*, p.96-104.

[121] Widrow, B. & Lehr, M.A., 1990. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9), p.1415-1442.

[122] Williams-Hayes, P.S., 2005. Flight Test Implementation of a Second Generation Intelligent Flight Control System. Technical Report NASA/TM-2005-213669.

[123] Xie, X. & Seung, S., 2004. Learning in neural networks by reinforcement of irregular spiking. *Phys. Rev. E*, 69, 041909.

[124] Zhang, J.S., Hu, H.X. & Nakamura, S., 2006. Automatic derivation of a phoneme set with tone information for Chinese speech recognition based on

mutual information criterion. *IEEE International Conference on Acoustics, Speech and Signal Processing, 2006*. Toulouse, France, May 2006, vol. 5, pp. 45-48.

[125] Zhang, L., Tao, H., Holt, C., Harris, W.A. & Poo, M.M., 1998. A critical window for cooperation and competition among developing retinotectal synapses. *Nature* 395, p.37-44.