Neural Pipelines: For the Co-ordination of Activity in a Multi-layered Neural Network

Rebecca Frances Naylor

Submitted for the Degree of Doctor of Philosophy

University of York

Department of Computer Science

December 2011

Abstract

The ‘Neural Pipeline’ is introduced as an artificial neural network architecture that controls information flow using its own connection structure. The architecture is multi-layered with ‘external’ connections between the layers to control the data. Excitatory connections transfer data from each layer to the next and inhibitory feedback connections run from each layer to the previous layer. Using these connections a layer can temporarily silence the previous layer and stop further inputs until it finishes processing.

When excitation and inhibition are balanced, waves of activity propagate sequentially through the layers after each input; this is ‘correct’ behaviour. When the system is ‘over’ inhibited, the inhibitory feedback outweighs the excitation from the input. At least one layer remains inhibited for too long so further inputs cannot stimulate the layer. Over inhibition can be corrected by increasing the delay between inputs. When the system is ‘under’ inhibited the excitation in the layer is larger than the inhibition. The layer is therefore not silenced and continues to spike.

In the layers, excitatory and inhibitory spiking neurons are randomly interconnected. Changing layer parameters influences the system behaviour. Recommendations for correct behaviour include: low neuron connectivity and balancing the external inhibition and layer activity. With variations of only the internal topology and weights, all three behaviours can be exhibited.

Each layer is trained as a separate Liquid State Machine, with readout neurons trained to respond to a particular input. A set of six shapes can be learnt by all layers of a three layer Neural Pipeline. The layers are trained to recognise different features; layer 1 recognising the position while layer 2 identifies the shape. The system can cope when the same noisy signal is applied to all inputs, but begins to make mistakes when different noise is applied to each input neuron.

The thesis introduces and develops the Neural Pipeline architecture to provide a platform for further work.

Contents

List of Figures

List of Tables

Acknowledgements

With thanks to Dr Simon O'Keefe and Prof Jim Austin for supervising the work, for their time and advice.

The suggestions and feedback from the Active Vision Group have been both interesting and helpful. Thank you to Prof Netta Cohen, Dr Marc de Kamps and Dave Harrison of the University of Leeds and to Prof Kevin Gurney, Dr Tom Stafford and Donny Yates of the University of Sheffield.

YCCSA is a rewarding and interesting environment to work in, so thank you to the members of YCCSA.

Thank you also to the White Rose Consortium who have supported this work with a White Rose Studentship.

Finally thank you to my family, particularly to Adam, for support throughout.

Declaration

The work presented in this thesis is my own work with the exception of works that are attributed and cited to other authors.

The document is formatted using Olivier Commowick's thesis document class for LaTeX. This can be found at http://olivier.commowick.org/thesis_template.php.

Elements of the work that appears in chapters 3 and 6 have been presented before in the following paper:

Rebecca Naylor, Simon O'Keefe, Jim Austin and Netta Cohen, 'Coordination of multi-layered neural computation - a Neural Pipeline approach', AISB'11, 2011.

Chapter 1

Introduction

1.1 Background

Our brains perform computational tasks using completely different hardware to traditional computer systems. Artificial neural networks are computer architectures inspired by the structure of brains. The aim of such biologically inspired computation is to borrow and simplify biological components where they may be useful while remaining free from the limitations imposed upon biological systems.

Artificial neural networks use the same high level concepts as biological neural networks. They both use interconnected groups of neural cells that each have a changeable internal state. The state is determined by the cell inputs. When the state reaches a specified threshold value the cell produces an output. The models that are most similar to biology produce spikes, this type of model is used in this work. When considered as an entire system a biological neural network and an artificial neural network have the same high level properties. They both represent some form of memory, in that they provide certain responses to different input stimuli. Both types of system can learn, by adapting this memory and therefore their behaviour in different situations.

Timing of information is important for both traditional computers and for biological neural systems. In traditional computer systems pipelines can be used to perform this task. Synchronous pipelines use a global clock to keep the timing correct but there are problems with providing a synchronised clock pulse to every section of the pipeline. Asynchronous pipelines use a handshaking mechanism to remove the need for a global clock.

In biology there is no global clock or designer to control the order of information for each new task. The structure of the architecture in the brain must regulate the flow of information. Based on this assumption it should be possible to develop an artificial neural network architecture similar to an asynchronous computer pipeline, to regulate information flow. The Neural Pipeline architecture introduced in this thesis aims to do just this.

1.2 Aim

The primary aim of the thesis is to investigate the following hypothesis:

> 'Coordinating the activity of firing neurons represents a method of controlling behaviour in spiking neural network memory.'

In this context ‘coordinating the activity’ is taken to mean using the neurons to regulate the spiking of other neurons in the system. The term ‘controlling behaviour’ here means to regulate the timing of the system. A ‘firing neuron’ is a neuron that is actively spiking.

The secondary aim of the work is to identify how parameter choices such as the connections, their weights and delays can be used to influence coordination.

These aims will be achieved by developing a layered artificial neural network architecture. The architecture should coordinate activity across its layers using the activity produced by the system, without the need for a system clock. This should be achieved using the objectives outlined below.

1.2.1 Objectives

In order to achieve the aims a number of objectives are set out. These are listed below:

1. **Compare the different methods of achieving the aim and any similar work by surveying relevant literature.**

   The choices made for the architecture should be based on this review. It should consider the different possibilities for the components that will make up the structure of the architecture, for example different artificial neuron models. This will allow comparison and a selection of the most appropriate options. This objective is addressed in chapter 2.

2. **Model and simulate the architecture using suitable software.**

   The simulation should include tests of the influence of different system parameters on the system and situations in which different parameter settings are preferable. The parameters should be adjusted to improve the performance of the architecture under different conditions. To identify the impact of any given parameter, this parameter should be varied while fixing the others. This will expand understanding of the architecture and how it may be set up to perform particular tasks. This objective is addressed in chapters 3 and 4.

3. **Produce analysis of the architecture.**

   The analysis should help to gain an understanding of why the architecture behaves as it does. This analysis can be either experimental or mathematical. This objective is addressed in chapter 3.

4. **Train the system to recognise a set of different input patterns.**

   As the architecture to be developed is a neural network memory it should be tested using pattern recognition. The architecture should be able to correctly identify a number of different inputs. This objective is addressed in chapter 6.

5. **Evaluate the success of the architecture from a computational and a biological point of view.**

The computational success is considered to be more important than the biological success, because the system is designed as a computational architecture with biological inspiration. The architecture is considered to be successful biologically if it represents a biologically plausible architecture, that may be found in the brain. It is considered to be particularly successful if it is possible that the brain may use architectures such as this for coordination. Computationally the architecture is considered successful if it fulfils the descriptions outlined in the aim and in objective 4. The evaluation should also consider possible applications of the architecture, in addition to the pattern recognition specified in objective 4. This objective is addressed in chapter 7.

1.3 Thesis structure

The document is organised into seven chapters, beginning with this, the introduction chapter. This provides the motivation to the work that has been carried out. It also introduces the aim of the work and objectives used to achieve this aim.

Chapter 2 provides an overview of the literature background to the thesis. The literature is split into two parts: the background required for neural network architectures and alternative architectures that partially achieve the aim set out for the thesis. The background covers biological neural networks, the development of artificial neural networks, different types of network and neuron model and learning methods that are used to train the networks. The architectures are broadly categorised as timing architectures such as computer pipelines and memory architectures such as liquid state machines. When work in the following chapters relies on or refers to this literature a reference is given to the corresponding section in the literature chapter.

The Neural Pipeline architecture is introduced in chapter 3. The chapter introduces the structure of the architecture, split into the external and internal structure. The external connection structure defines the Neural Pipeline architecture although the parameters used on the connections can be varied. The chapter also describes the internal structure that has been used in this thesis, although the same architecture could be used with different internal structures. The three fundamental types of behaviour that a Neural Pipeline can exhibit are introduced.

Chapter 4 describes the preliminary results that were used to determine suitable parameter values for the architecture. These results also contribute to the analysis of the architecture in chapter 5. All of the parameters that have been investigated are discussed, with reasoning for which parameters have been chosen for the architecture.

Analysis of the Neural Pipeline architecture is provided in chapter 5. Two main features of the architecture are analysed. Firstly the factors that influence the three behaviour types are investigated. The second part of the analysis concerns the duration that a specific input remains recognisable to the system.

The results of training a Neural Pipeline using the principle of Liquid State Machines are shown in chapter 6. Three different experiments are presented, the first trains a Neural Pipeline to recognise different shapes. Each of the layers of the architecture performs the same task. The second experiment is an extension of this where layer 1 identifies the position of an input and layer 2 identifies the shape. Experiment 3 considers the impact of noise on the system input.

Chapters 3, 4, 5 and 6 are formatted with the work that has been done presented in the main body of the chapter. At the end of each chapter there is a discussion of this work. The discussion includes ideas for further work, these ideas are introduced throughout the chapters when first mentioning the work that could be extended. The further work section in the discussion provides a more detailed account of the suggested work. Each of these main chapters has a summary section to give a concise account of the chapter.

Conclusions drawn from the work are presented in chapter 7, these conclusions are based on the work presented in the rest of the thesis. This chapter draws together the ideas presented in the discussion sections of the earlier chapters. The conclusion chapter also includes an evaluation of the architecture and work in the thesis, based on the objectives outlined in section 1.2.1. The possibilities for future work outlined in the earlier chapters are summarised in broader categories. The final section of this chapter is a summary of the key findings of the thesis.

CHAPTER 2

Spiking ANNs: Inspiration and Operation

2.1 Introduction

The Neural Pipeline architecture has been developed in this thesis to fulfil the aims outlined in section 1.2. Before introducing the architecture, it is necessary first to introduce the different concepts that the architecture uses. The first part of this chapter provides an introduction to Artificial Neural Networks (ANNs), the biology that inspires them and how they can be used to construct computational systems. The second part describes alternative architectures that can be used to perform the same type of task as a Neural Pipeline and compares the strengths and limitations of these approaches.

From the very early examples of ANNs the motivation has been that biological nervous systems are valuable sources of inspiration for Computer Science. A brief introduction to the cells and connections that make up these biological systems is given to provide background. This leads into the development of ANNs and how they have become more complex. These ANNs use different neuron models, some very simple, others able to more closely represent the underlying biology. High level structures of network are described, in terms of feedforward and recurrent networks.

For ANNs to be used for computation they must be trained to produce a particular response when given certain inputs. Different training methods are introduced to illustrate how they can be achieved and why they may be preferable. Usually the network itself is trained using one of these methods, but it is possible to add additional 'readout' layers to the network instead. These readout layers are trained instead of the original network. This is the technique used in the field of reservoir computing and particularly Liquid State Machines (LSM).

The aim which lead to the development of the Neural Pipeline architecture is to investigate the use of neurons to control the timing of other neurons within a system. This constructs an internal timing system for the architecture, so that it can provide a response to a set of computational tasks in the correct order. For this reason the architectures introduced here are considered in terms of their timing structure, applicability to perform computational tasks and their biological inspiration.

The architectures that are compared are types of associative memory, computer pipelines, synfire chains and liquid state machines. Associative memory and liquid state machines are considered because the aim is to investigate coordination in a neural network memory. These are both ways of implementing memory which can be used to perform computational tasks. Associative memory uses traditional computing approaches but liquid state machines represent a biologically plausible method. These memory structures do not use inherent timing mechanisms. Computer pipelines and synfire chains are introduced because of their timing structures. Synfire chains are not used to perform computational tasks (on their own) and computer pipelines are not biologically inspired. A comparison table of the architectures is provided to give a summary of their differences. None of the architectures on their own match the aims set out for the Neural Pipeline, which is the reason that the architecture has been developed.

2.2 Biological Neurons

Artificial neural networks draw inspiration, to different degrees, from biological neurons. Typically they are not intended to replicate biology precisely but to use an interpretation of biological behaviour to provide computational function. There are several reasons for this, the complexity of a biological system is difficult to replicate and highly computationally expensive. It is possible to represent the overall behaviour without the same level of detail, for example learning and memory. An outline of biological neurons is provided as an introduction to the neuron models later in the chapter.

2.2.1 Neural cells

Although neural cells were observed as early as the 1830s, the complete structure of neurons was only discovered in the 1950s with the use of electron microscopes [58]. It is this structure that is useful when determining the function of the neural cells.

An example of a neuron structure is shown in figure 2.1. This is a pyramidal cell, one of the most common neurons in the brain. It is composed of a number of dendrites, a cell body and an axon. The dendrites provide inputs to the neuron from other neurons and the axon provides the output. The majority of neurons communicate using a change in voltage, these changes are known as pulses or spikes. The cell body has electrical potential and this changes in response to input spikes from its dendrites. The cell body triggers a spike when enough inputs appear (within a certain time) at the dendrites, the spike travels along the axon toward the connected neurons. As axons are comparatively long and thin they have the same characteristics as electronic transmission lines.

Each of the neurons themselves has a complex internal structure. The neurons are composed of many components and each has a specific function. The most important with regard to artificial neural networks is the cell membrane. The membrane has capacitance and an electrical potential when compared to the inside of the neuron, this is necessary for the neuron to generate spikes. Other organelles control cell maintenance and development, digestion, energy production and protein synthesis [58]. These types of property are not necessary in artificial neural networks.

Brains have different types of neurons and also different types of cell used to support and repair the neurons. The different neurons can be identified by differences in their physical features, for example the presence of a long axon or in some no axon

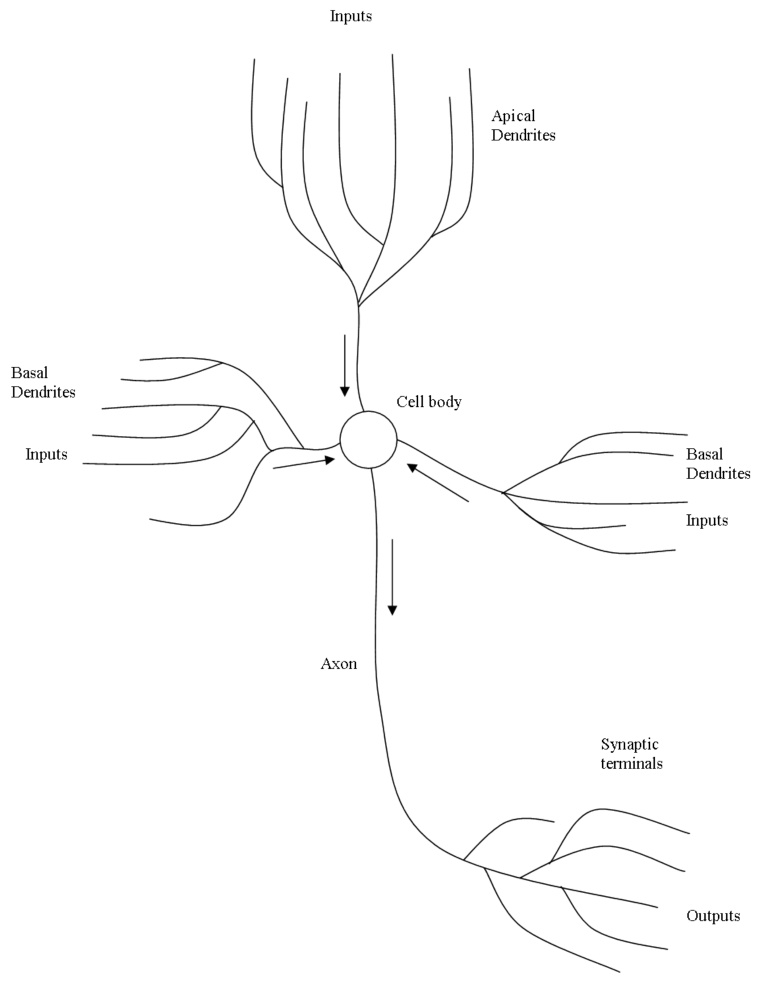


Figure 2.1: A pyramidal cell neuron based on a diagram in [28].

at all. The other cells are known as glial cells and 'nerve sheaths'. There are different types of glial cell with properties including the reception of neurotransmitters and release of other neurochemicals. The cells are thought to aid with the operation and development of synapses this allows them to increase plasticity in the brain [9]. Other roles are to influence blood flow [9] as well as repairing neurons and providing structural support [58].

Nerve sheaths are provided by the neuroglia, they surround long axons. They provide protection for the axons and can modify axons that must carry long distance signals. [58]. Artificial networks are not generally concerned with this level of detail.

2.2.2 Neural connections

The connections between neurons in a network are known as synapses, taken from the Greek word for connect. As in artificial networks the overall behaviour of synapses is to pass information only in one direction, from the outputs of one neuron into the dendrites of others. On a small scale the mechanism is more complicated, but this need not apply to artificial networks.

There are three stages in interaction between nerve cells; presynaptic, postsynaptic, and intervening. As the names suggest the presynaptic process involves the neuron before the synapse, the postsynaptic the neuron after the synapse and intervening between the two.

In real networks the positioning or 'relatedness' of the cells is important unlike neurons in artificial networks which are either connected or not. In real neurons the connectivity is less black and white. Neurons are closely packed together so they can often influence other neurons even when not directly connected. When the membranes of multiple neurons are 'next to' each other they can cause interference, similar to the concept of crosstalk in electronic transmission lines.

The nearest contact is when the cell membranes are in contact with one another. One situation where the membranes are in contact is a 'gap junction', this is known as an electrical synapse because it allows electrical signals to pass between neurons. Electrical synapses are able to respond more quickly than the alternative chemical type of synapse. This is because they do not use transmitter chemicals. Electrical synapses are considered to be important in synchronisation [1].

There are comparatively few electrical synapses in the brains of mammals. The majority of synaptic connections are chemical synapses. They take an electrical input spike from a presynaptic neuron and convert this into an electrical output using a 'neurotransmitter' chemical that travels across the synapse [58]. There are many different types of neurotransmitter chemical and their behaviour is complex. Most artificial neural networks use only their main features, the strength of response and whether the connection is excitatory or inhibitory. If the synapse is inhibitory it decreases the response of the postsynaptic neuron, excitatory connections increase the response [63].

There are also different ways to connect synapses. Single connections can be formed from axon to axon (axoaxonic), dendrite to dendrite (dendrodendritic), axon to dendrite (axodendritic) or axon to cell body (axosomantic). Often synapses are

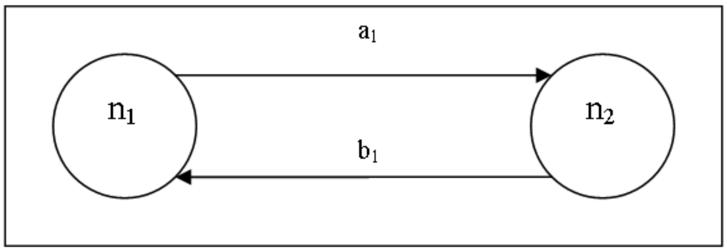


Figure 2.2: A reciprocal pair between two neurons.

grouped together, when they are in the same direction they are all axodendritic. If they form a path from one neuron to another, then from that neuron to a third they are called 'serial synapses'. It is also possible for there to be a synapse between one neuron and another and a second synapse connecting from the second neuron to the first. If they are together they are considered to be a 'reciprocal pair' (shown in figure 2.2) and if they are separate a 'reciprocal arrangement' [58].

Just like electrical circuits biological systems tend to have similar units that connect together to produce more complex behaviour. In biology the lowest level is formed using synapses connected to a particular point such as an axon or a cell body. These low levels are 'microcircuits' and are often repeated within this layer, as for example electronic logic gates could be used to make higher components. The next level, a 'local circuit', provides longer distance connections using a dendritic branch or axon. It is contained in one region. Connecting different regions occurs at the next level. Higher still the connections can pass through several regions. The highest level, a 'distributed system', is considered to be a connection between regions that will cause behaviour that involves the entire system. Changes can be made to the system by altering weights of the synapses or by constructing new connections between neurons [28].

2.2.3 Dale's Principle

The use of Dale's principle in artificial neural networks is an interesting example of biological grounding. Dale's principle states that every synapse from a given neuron uses the same neurotransmitter or set of neurotransmitters [20]. This has the result that the outputs from any particular neuron cannot be both excitatory and inhibitory. There are only a few known exceptions to this in biology so it is a good general rule.

Most artificial neural networks do not follow Dale's principle, although it has been considered in some. The resulting networks are generally known as 'sign-constrained'. Abbott [2] splits from Dale's principle the idea of sign changes in synaptic weights when learning. Most learning algorithms allow a given synaptic weight to change sign as well as to contradict Dale's principle. Certain implementations of networks impose only a constraint to stop sign changes [17], while others implement Dale's principle fully [59]. When the networks learn new information there is a mechanism in place to prevent Dale's principle being contravened. This can be achieved either by ignoring proposed changes to the synaptic weights which would change sign or setting them to zero as discussed in [4].

2.3 Development of the Artificial Neural Network

Artificial Neural Networks are a surprisingly long established field, the earliest work is usually traced back to the 1940s with the advent of the first electronic computers (for example in [29, 28]). People were considering how groups of neurons may interact as far back as Bain in the 1870s. He suggested that it would be possible for groups of neurons to be used to store multiple memories [68]. Artificial neural networks use terminology originally used for biological networks, the nodes being 'neurons' and the connections 'synapses'. Both of these terms were introduced in the 1890s, neuron slightly pre-dating synapse because neural cells were discovered before their interconnections.

There was a good deal of progress in the field through the 1950s and 60s. New structures of network were created such as the perceptron introduced in section 2.3.1.1. Associative memories (section 2.8) were introduced in this period, an example of which is the Learning matrix described in section 2.8.2. Different methods of training the networks were introduced, such as the Widrow Hoff delta rule (section 2.5.2). ANN hardware was commercialised in this era with the foundation of Adaptronics Corp.

In 1969 Minsky and Papert published a paper that is believed by some to have caused the slowdown in the research into neural networks that occurred for around a decade after this [38]. In it they proved the limitations of single layer perceptrons and cast doubt on the viability of training multi-layer perceptron networks. Other factors such as a lack of computing power may have influenced this slowdown [28].

The 1980s showed a regained interest in neural networks research with key developments by many researchers including Hopfield, Kohonen and Rumelhart. Hopfield created Hopfield networks (section 2.8.1) that use feedback from the outputs. Kohonen expanded work on unsupervised (self-organising) networks while Rumelhart developed back-propagation as a new learning method. These developments helped to overcome the limitations of the perceptron architecture.

In the following decades there has been further expansion of the field, particularly with the development of spiking neuron models. These models are more similar to the underlying biology, they are discussed further in section 2.4.

The field has developed greatly over time, although many of the original ideas are still used today. Maass [46] splits the different types of ANN into three generations. They progress from the first generation of digital neurons to the more biologically realistic third generation of spiking neurons. The generations relate to when each type was developed, but there are still reasons to use the earlier generations today. These networks tend to be less computationally expensive than their spiking counterparts.

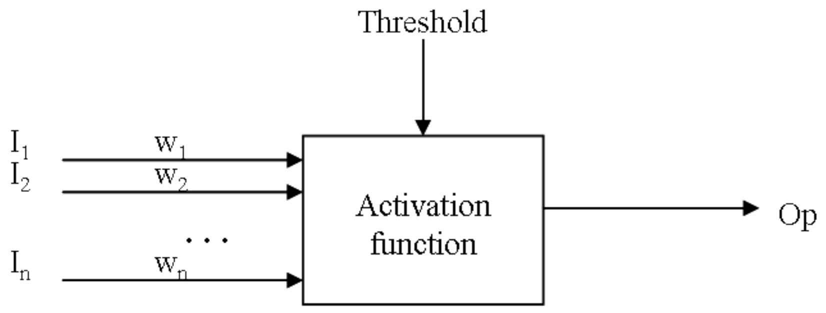


Figure 2.3: Generic neuron model. Inputs are fed into an activation function that uses the threshold to determine its output.

A neuron model of any generation produces an output from a weighted sum of inputs using an 'activation function'. The activation function is the function that controls the response of the neuron depending on the value on its inputs. A threshold value is used as a parameter for the activation function to determine its output. All of the neuron models can be represented using the simple diagram shown in figure 2.3.

2.3.1 First generation

First generation neural networks use McCulloch-Pitts neurons. McCulloch and Pitts constructed this binary neuron model (as shown in figure 2.4) in 1943 [54]. These neurons produce a digital output of 0 or 1 depending on their input. If the sum of their weighted inputs is greater than a given threshold the output is 1 otherwise it is 0. For example with a threshold of 0, this means that their activation function is a step function. The step function is simply 0 for negative outputs and 1 for positive and zero outputs. Thus it can be expressed as:

$$s(x) = \begin{cases} 1, \text{if } x \geq 0 \\ 0, \text{if } x < 0 \end{cases}$$

A diagram of such a neuron is shown in figure 2.4. The threshold is represented by t while the input weights are denoted by w. Where the weights w are positive the inputs are excitatory and where they are negative the inputs are inhibitory.

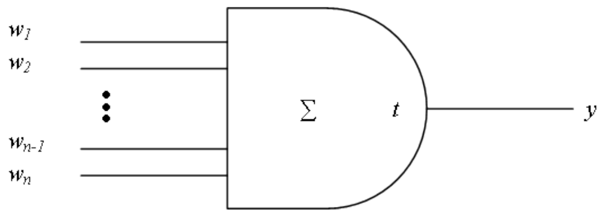


Figure 2.4: A McCulloch-Pitts neuron.

These individual neuron models are combined to form first generation networks.

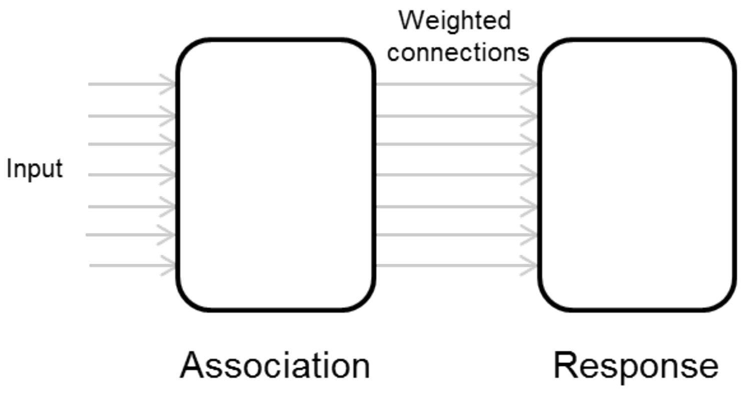


Figure 2.5: The perceptron structure based on a diagram in [54].

They include perceptrons and multi-layer perceptrons (described in the following section) and Hopfield networks (section 2.8.1). A key characteristic of first generation neural networks is that they are only able to provide digital outputs. They use thresholding functions to provide their output.

A possible advantage over the next generations is their relative simplicity for implementation and understanding. Disadvantages are their lack of biological realism and related lack of complexity in behaviour.

2.3.1.1 Perceptrons

The perceptron is an example of a first generation structure. Rosenblatt first introduced the perceptron in [57]. The structure of a single layer perceptron is shown in figure 2.5. Although it appears to have two layers, it is considered to be a 'single layer perceptron' because only the response layer can be trained. This is due to the changeable weight values on its connections. This means that people consider this response layer to be the single layer of importance.

Rosenblatt used the structure for image recognition so the inputs are provided by a retina grid. The association layer is made up of association cells, each of which is the equivalent of a McCulloch-Pitts neuron (figure 2.4) and has a number of inputs. These inputs are randomly chosen from the retina to split up the input. The connections to the response cells can be trained using their weight values. There is also feedback between this response layer and the association layer. Each node uses feedback to encourage its own inputs, or to inhibit inputs to the other output nodes. This means that only one output is active at a given time.

Single layer perceptrons can only handle linearly separable functions [54]. Instead non-linearly separable functions can be handled using an extension to the perceptron architecture: multi-layer perceptrons. An example of a multi-layer perceptron architecture can be seen in figure 2.6. The development and successful training of the multi-layered perceptron was important in the revival of the field of neural networks in the 1980s (as described in section 2.3). The networks tend to be fully connected and can be extended using multiple hidden layers.

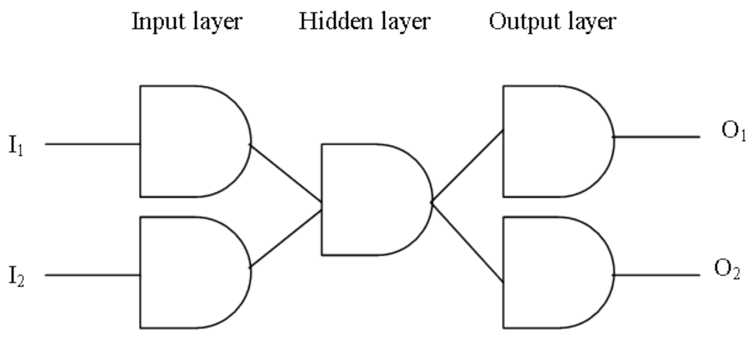


Figure 2.6: A three layer perceptron architecture based on an example in [54].

2.3.2 Second generation

Second generation neural networks differ from first generation networks as they can produce analogue outputs and can handle analogue inputs. They use a continuous activation function, rather than the thresholding step function of first generation networks, to cope with the continuous analogue input. Examples of continuous activation functions include the piecewise linear function and the sigmoid function. The piecewise linear function represents a sloping increase between negative and positive values, rather than a discrete step. The length of the slope can be altered by changing the 'amplification factor', if the amplification factor is set to be infinite the piecewise linear function is the same as a step function [28] . Sigmoid means 's-shaped' [11], and an s-shaped sigmoid function represents a continuous transition between positive and negative values. In neural networks it is the most frequently used activation function [28]. As with the piecewise function it is possible to alter the shape by altering a parameter of the function. An advantage over the step function is that the sigmoid function is continuously differentiable.

Second generation neural networks represent a step closer to biology when compared with the first generation of neural networks. This is an advantage when trying to model biological behaviour.

2.3.3 Third generation

Spiking neural networks are the most similar to biological networks of neurons. Individual spikes are used to represent the inputs to and outputs from neurons. Information can be encoded using the timing of the spikes.

The neurons in spiking networks are different to the earlier generations because they must handle spike trains. A simple spiking neuron is the 'integrate and fire' model (section 2.4.1), more specifically leaky integrate and fire. Integrate and fire neurons become more active as more spikes are received, the leaky component means that their activity decreases over time. The neurons fire when their activity reaches a certain level and often a lag is introduced so that their activity does not increase for a time after the neuron has fired.

An advantage of third generation networks is their improved biological accuracy. This is of particular relevance to this work. Another advantage is that it is possible to represent networks of previous generations using them [46]. The main disadvantage of the third generation networks is their increased complexity. As computational power improves, the simulation of such networks becomes easier.

2.4 Spiking Neuron Models

There are various different types of spiking neuron model with different levels of complexity. All of the neurons respond to inputs by producing an output spike, but they use different equations to implement this. An overview of three types are given below in order to compare them, to show why one type may be chosen over another. The models described here are the commonly used integrate and fire neuron, the more biologically realistic Hodgkin-Huxley neuron and the Izhikevich neuron which attempts to combine the best aspects of the other two. There are alternatives and variations of these models. A review of different models is carried out by Long and Fang in [44].

2.4.1 Leaky integrate and Fire

A simple spiking neuron model is the 'leaky integrate and fire' model. This is the equivalent of taking the weighted sum of inputs with a spiking input. The neuron can be modelled as an electric circuit with one resistor R in parallel with a capacitor C the input to this circuit is a current *I(t)*.

Equation 2.1 can be produced from this circuit [47] with u representing the voltage across C.

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt} \tag{2.1}$$

It can be rewritten as equation 2.2. Here u is called the membrane potential with τ_m introduced as the time constant of the membrane potential. The influence of each input spike is incorporated into the internal state of the neuron. The leaky aspect is that the internal state steadily decreases, this has the effect of reducing the likelihood of firing after periods of inactivity.

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \tag{2.2}$$

The differential equation 2.2 does not fully describe the neuron, there is the additional aspect of a threshold. When the membrane potential u crosses the defined threshold the neuron will spike. After this spike the membrane potential is reset to a resting potential.

The input *I(t)* can either be a constant current, a varying current or a series of input spikes. The graphs in figure 2.7 demonstrate how a single LIF neuron responds to either a constant input (graph a) or a varying input (graph b). The constant input current produces a periodic response, with the neuron spiking when it reaches the threshold of 1. With a varying input current the membrane potential rises irregularly until it reaches the threshold. The response to a spiking input would be similar to that of graph (b).

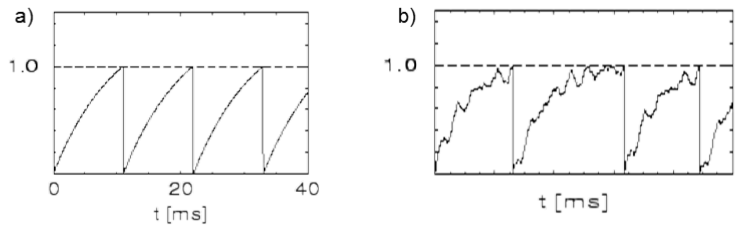


Figure 2.7: Graphs from [23] showing the membrane potential response of a single LIF neuron to (a) a constant input current and (b) a varying input current.

2.4.2 Hodgkin Huxley

The Hodgkin-Huxley equations are based on biological results from squid neurons [63]. This means that they are more biologically accurate than the integrate and fire model. They are four differential equations, the equations themselves can be found in [63].

To cause a spike to be produced two or more voltage dependent ion channels and one fixed channel are needed. One voltage dependent channel causes the increase in membrane potential, the other causes the decrease. In neurons the two voltage dependent channels are sodium and potassium and the fixed one is also for potassium. The fixed potassium channel maintains the resting potential, by allowing potassium to leave the cell. When the ion channels controlled by the neurotransmitters change the membrane potential the voltage dependent sodium ion channel opens. This causes the membrane potential to rise. The sodium channel then becomes inactive, and the potassium channel opens, causing the membrane potential to fall down to below the resting potential. Then both voltage dependent channels close and the potential returns to the resting value [63].

The Hodgkin-Huxley equations describe this channel behaviour. The three equations that represent the activation and inactivation or the sodium channel and the activation of the potassium channel all use the same formula. The fourth equation gives the capacitance of the neuron.

The potassium channel reducing the potential to below the resting potential means that there is a 'turn around' time before another spike can be generated. This limits the frequency of firing.

When compared to the integrate and fire model the Hodgkin-Huxley equations have the advantage of being more biologically accurate. However they are more computationally expensive. Izhikevich [35] constructs a model that he claims has the advantages of both.

2.4.3 Izhikevich

Izhikevich [35] simplifies the Hodgkin-Huxley equations into two equations and tests their ability to model the biology by attempting to represent different types of neuron with different properties. One such property is bursting. A burst of spikes is considered to be a number of spikes received over a short time period, with a particular pattern [36]. Bursting is thought to strengthen signals, because it provides repetition. In addition to this Izhikevich suggests that the frequency of the spikes within a burst can allow certain neurons to be targeted [36]. This occurs when the interspike frequency within a burst is the same as the resonant frequency of the neurons. This allows a preference for certain neurons without having to alter the synaptic weights (as discussed in section 2.5).

Different types of neuron have different characteristic patterns of firing. The most common is ‘regular spiking’, which settle into a periodic spike train after an initial period with more frequent spikes. The period is slower than the fastest possible time. ‘Intrinsically bursting’ have more frequent spikes in the initial period and ‘chattering’ have regular periods of bursts of spikes. The inhibitory patterns are ‘fast spiking’ or ‘low threshold’ spiking. Fast spiking is the same as regular spiking with a higher frequency and a very short initial period. Low threshold spiking as suggested by their name have a smaller threshold. In [35] Izhikevich shows that the model is able to provide the responses of each of these different types of neuron, so one of the strengths of the model is that it can represent complex biology.

The computational efficiency is indicated by the simplicity of the equations when compared to the Hodgkin-Huxley and the ability to represent a large network of neurons on an outdated computer. A possible criticism is the lack of direct comparison with either Hodgkin-Huxley or Integrate and Fire neuron models in [35].

2.5 Learning

Learning is the process of the network adapting in response to the inputs that it is given in order to perform the desired task. This section introduces different methods used for training a network and some of the rules that can be used for this training.

2.5.1 Learning Methods

There are three different methods for training a network so that it learns how to respond. These three types are supervised, reinforcement and unsupervised learning. They relate to the amount of prior knowledge the system is given about the data it will be classifying. Supervised means that the system is trained to recognise a set of predefined patterns, this is useful if all of the types of input pattern are known beforehand. Training on a Neural Pipeline in this thesis is supervised for this reason. If the categories of pattern are unknown then the system can determine classes by itself, this is unsupervised learning. Reinforcement learning lies between the two of these approaches. All three are described below to give a comparison of when they are best used.

Supervised learning

In supervised learning the desired output of the system is known for a particular input. The input vector is presented to the inputs of the network and the desired output is compared with the actual result from the system. This comparison provides a measure of the error which is then fed back into the system and used to adjust the weight values to create an output closer to the desired one. The process is repeated until the weight values allow the network to produce a value that matches the desired output. It is done for the different inputs and outputs that the system is to be trained to recognise. Supervised learning uses a closed loop feedback system, and can be considered to be an optimisation problem. The weights must be set to give the optimum output for all of the inputs encountered. In supervised learning the system can have gradient information which makes this optimisation easier [28].

Reinforcement learning

In reinforcement learning the desired outputs for input patterns are not provided for the system. Instead the neural network is told whether the output it has produced is correct or incorrect. Learning takes place using a measure of 'reward' with the areas that provide a correct output having their weights strengthened. A penalty is applied to the areas that provide an incorrect output by decreasing their weights [53]. The name reinforcement is important because the system behaviour is reinforced by continuous input from the network environment.

An advantage of reinforcement learning is that the system is able to adapt online [28]. A disadvantage compared to supervised learning is that it is more difficult to implement. It is used less frequently than either supervised or unsupervised learning [53].

Unsupervised learning

In unsupervised learning example input patterns are presented to the network, but no matching outputs are given. It differs from reinforcement learning because there is no feedback to tell the network whether the output is correct or not. The network must determine characteristics of the input patterns itself in order to classify them. An advantage of this compared to the other forms of learning is that it should not need setting up depending on the particular task. A disadvantage is that it is more complicated to implement.

2.5.2 Learning Rules

There are many different rules used to train neural networks. They can be broken down into different categories including gradient descent, Hebbian, competitive and stochastic. A brief description of each of these types is given below.

- **Gradient Descent** rules minimise an error function. The network weights are changed by comparing the actual output of the network with the desired output. They are updated at a specified learning rate which controls how much the weight can change with each update. The Delta rule and backpropagation are both Gradient Descent methods.

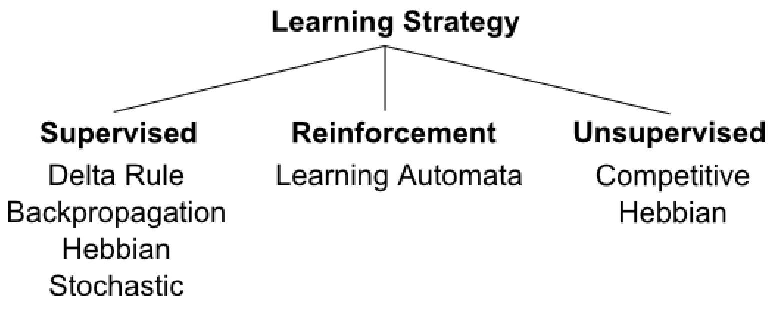


Figure 2.8: Learning rules categorised by the learning method. This figure is from chapter 2 of [53].

- **Hebbian** learning strengthens connection weights when the neurons at either side of the connection fire simultaneously.
- **Competitive** learning has neurons compete to represent a particular input. The weights are changed so that the most active neurons have their weights increased. Winner takes all is an example of competitive learning in which only one neuron represents the input.
- **Stochastic** learning uses a probability distribution to alter the weight values. An example of a stochastic learning rule is Simulated Annealing.

These rules are related to the learning method used. Some are specifically used for supervised learning, others unsupervised and some for both. Gradient descent methods are used for supervised learning, Competitive for unsupervised and Hebbian can be used for either. This is shown in figure 2.8 from chapter 2 of [53].

Two of these learning rules are explored in more detail. The earliest learning method, Hebbian learning, is introduced as it is used in a number of the architectures described later in the chapter. The delta rule is used for training the Neural Pipeline architecture in chapter 6. There are other ways to perform learning in an artificial neural network, such as backpropagation which extends the Delta rule so that it is suitable for the multi-layer perceptron (see section 2.3.1.1). More details about backpropagation can be found in [54].

Hebbian Learning

Hebb proposed a learning method, now known as Hebbian learning, in 1949. As a psychologist his work was based on real neurons, but the same principles can be applied to neuron models. He proposed a method of associating the physical process of neurons firing with creating memories, this is the basis of learning in a neural network.

Hebb's rule is that a synapse should be strengthened if the neurons on both sides of it are active at the same time (as shown in figure 2.9 i). These neurons are known as pre-synaptic and post-synaptic neurons. This process allows a network to become specialised towards certain patterns, as the strengthened synapse increases

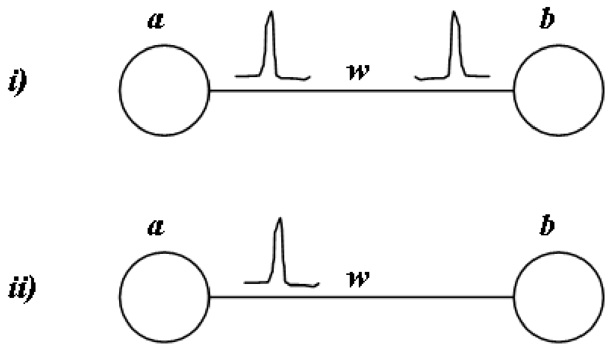


Figure 2.9: Hebbian learning in situation i) an increase in weight w occurs and in situation ii) a decrease occurs.

the likelihood of the neurons firing at the same time which in turn strengthens the synapse [38].

An extension to the original rule permits decreases too, because increases alone can lead to errors. An example of such an error can be found in chapter 2 of [54]. Decreases occur when a neuron at either side is active when the other is not (as shown in figure 2.9 ii).

Haykin [28] provides a list of the four mechanisms that define a 'Hebbian synapse'.

1. **Time dependency**: Hebbian synapses are time dependent because their state is changed due to the precise timing of the input signals

2. **Local**: The synapse makes use of local information in order to change its state.

3. **Interactive**: The change in the synapse depends on the interaction between both of its input signals, neither one on its own can be used to predict how the weight should change.

4. **Conjunctional or correlational**: A Hebbian synapse may be thought of as conjunctional because the conjunction of input signals triggers a weight change. It can also be thought of as correlational because the input signals become correlated over some time period in order to change the weight.

In the same way that a system can be Hebbian, it is possible for it to be anti-Hebbian or non-Hebbian. Anti-Hebbian is the opposite of Hebbian, in that signals which arrive together decrease the associated weight and signals that arrive independently increase the weight. Non-Hebbian does not exhibit Hebbian behaviour at all.

Delta rule

The delta rule was introduced by Widrow and Hoff in 1960. It is more flexible than Hebbian learning as described above, because it uses an error value to decide by how much to alter the weights. The error is the difference between the required output and the actual response. With each weight adjustment the actual output is brought closer to the desired output. This is known as gradient descent. So for example in a binary network if the actual output value is 1 and the expected output is 0 then the weight is too high, so it is reduced.

The rule used for performing the update is given in equation 2.3. Δw is the change to be applied to the weight. The term $D_{op} - C_{op}$ is the error, the desired output D_{op} minus the current output C_{op}. The size of the change is dictated by the learning rate r. This controls how quickly the weight is able to converge on a value that will provide the correct output.

$$\Delta w = r(D_{op} - C_{op}) \tag{2.3}$$

This rule is applied to each of the input connections of each of the neurons in the network. For a single pattern the rule can be used in this format. To store multiple patterns it is run for each of the patterns in turn. Details of how the rule was used in Widrow and Hoff's ADALINE network can be found in [54].

2.6 Structures of neural network

Artificial neural networks can be split broadly into two types based on the direction of information flow through the system. These types are feedforward where the information flows in only one direction and recurrent where there is feedback in the system. The Synfire Chain introduced later in section 2.10 is an example of a feedforward network and the Hopfield network (2.8.1) is an example of a recurrent system. The Neural Pipeline architecture developed in this thesis is a recurrent architecture. The descriptions below describe these structures in more detail.

2.6.1 Feedforward Neural networks

Feedforward refers to the direction in which the data travels through the network. In feedforward systems the data is only passed forwards, there is no feedback. Examples of feedforward network are single or multiple layer perceptrons. In a single layer system data is passed from the input nodes to the output nodes. In a multiple layer system there can be any number of hidden layers between input and output, but data always progresses from one layer to the next. Figure 2.6 illustrates a multiple layer feedforward network. Another example of a feedforward network is that used in [64]. They used this network for the purpose of binding individual words into sentences and found that it performed this task.

2.6.2 Recurrent neural networks

Recurrent neural networks include the concept of feedback. Data is passed not only from input to output nodes, but is passed back from the output to the input. Recurrent neural networks can be single or multiple layer. A specific example of a recurrent network is the Hopfield network (section 2.8.1)

When directly compared for prediction both [18] and [10] found feedforward networks to outperform recurrent. In a comparison for pattern recognition [65] the recurrent network outperformed the feedforward network. This suggests that different architectures are useful for different tasks.

Advantages of feedforward networks include their relative simplicity. However recurrent networks are able to exhibit dynamic behaviour, because of their feedback loops. This means that they are able to change and refine their value using the additional information provided by the feedback connections. It allows them to produce more complex behaviour than feedforward networks, but also means that they must settle onto a value before the output is taken.

Other examples of recurrent neural networks of particular interest, because of their use of reciprocal connections, are bidirectional associative memories (BAM) and recirculation networks. Bidirectional memories use the same synaptic weights for both sides of each reciprocal pair, but recirculation networks can use different weights. An example use of a recirculation network is for face recognition in [16]. Bryliuk et al [16] found that the network was useful for extracting image features and recognising images successfully, this means that recirculation networks are of interest for vision.

2.7 Architectures for Memory and Coordination

The following sections outline architectures that address different aspects required to achieve the primary aim of this work (set out in section 1.2). Two different types of architecture are presented, architectures used for coordinating activity and memory architectures for storing information used to perform a computational task. As the aim is to investigate coordination, it is clear why architectures developed for this purpose are presented. The aim specifies that the system should provide coordination in a neural network memory so different memory architectures are introduced for comparison.

The memory architectures are presented first. They are split into associative memory and reservoir computers. Associative memory, as the name suggests are used to associate a particular input pattern with a given output response. Reservoir computers use randomly connected networks to form a fading memory. The input patterns cause particular perturbations in the network and these perturbations can be associated with a given output.

Computer pipelines and Synfire chains are architectures that coordinate activity. Computer pipelines provide a mechanism of splitting a large processing task into several smaller ones in order to speed up the overall throughput. In a pipeline it is important that the subtasks are coordinated so that they receive the correct data to process at the right time. Synfire chains are neural architecture that coordinate activity, they are found in biological neural networks. They are composed of a number of layers with feedforward connections between them. Activity passes through each layer in turn so the architecture can be used as a timing structure.

A comparison of each of the architectures is given in section 2.12. The strengths and weaknesses of the different architectures are given, with regard to the aims of this thesis. Particular attention is given to the timing structure, suitability for performing a series of computational tasks and their biological plausibility. The table demonstrates that no single architecture in the literature achieves the required aims for the Neural Pipeline architecture.

2.8 Associative Memory

The first category of architectures introduced for comparison with the Neural Pipeline architecture is associative memories. These memories provide a location in which to store the information that a system learns. The information must be retrieved from the memory to produce a usable system. Computational memories are similar to their biological counterparts in these respects, but they use a different storage method.

Associative memories allow an input to be associated with a particular output pattern. When the system learning is supervised the input pattern can be presented to a network at the same time as the desired output pattern. When the output pattern differs from the input it is known as 'hetero-associative', and when they are the same 'auto-associative'. Hetero-associative systems do not even need to have inputs and outputs of the same type, so an input sound (for example birdsong) could produce a pictorial output (a picture of the type of bird).

The concept of associative memory is important for pattern recognition because it allows generalisation. That is to say that an imperfect input, for example corrupted with noise, can be correctly classified.

Associative networks are given the same two main classes that are applied to neural networks more generally; feedforward and recurrent [29]. Feedforward have a single layer of nodes with separate inputs and outputs whereas recurrent networks have feedback in terms of their outputs being connected back to their inputs. The Hopfield network is introduced as an example of a recurrent network.

The Learning Matrix and CMMs are both types of associative memories that use matrices. The Learning Matrix provides background for CMMs, and CMMs considered to be a way of implementing the Neural Pipeline in hardware in the future.

2.8.1 Hopfield Network

Hopfield networks are recurrent, they have a single layer of neurons and the outputs of each neuron are connected to the inputs of every other neuron. Hopfield showed that the network would converge in this instance [32]. There are instances of the Hopfield network with both excitatory and inhibitory 'self-connections' for example Li [43] shows that a network with positive feedback will converge and from experimentation could perform better than the original.

Hopfield networks are auto-associative [55], the required output is the same as the given input. This could seem like an unlikely requirement for a system, but it is important because the system can be used to handle imperfect versions of the input. An example would be noise corrupted inputs could be restored using the system.

The network is composed of a single layer of neurons. It uses feedback by connecting each output back to the inputs of each neuron except itself. Each of these neurons takes the sum of its weighted inputs, if this is over a given threshold then the neuron fires.

Hopfield networks are represented using values of +1 and -1 rather than the more traditional values of 1 and 0. This is because Hopfield networks are recurrent and the use of only positive (and 0) values would cause positive feedback. This is undesirable because it would stop the system from stabilising on an output value. The use of +1 and -1 also means that it is possible to use a fixed cut-off of 0 on the output to find the result.

A strength of the Hopfield Network is that, because it is recurrent, it can handle non-linearly separable data.

A limitation of Hopfield networks is that they must stabilise on an output value. The feedback in the system means that the initial value is not necessarily the correct response. The threshold required depends on the set of patterns that is to be learnt, [54] describes how this threshold can be found.

The desired outputs should be in stable states. A stable state is considered to be one in which the system will finish when all of the outputs have finished updating. If there are more stable states than classes then it is possible that the stable state entered will be an unknown state, this can be rectified by adding more neurons to the network [54].

Another limitation of Hopfield networks is a low storage capacity. In [5] Amit et al find the theoretical maximum number of patterns that can be stored to be 0.138 times the number of neurons in the network.

2.8.2 Learning Matrix

The learning matrix represents an early example of associative memory, Steinbuch conceived the network in 1958. An advantage of the system is that it can be realised as a physical device; a matrix with the inputs on the horizontal x lines, and the outputs on the vertical y lines. There are connections with associated weights between these lines, and in the physical device these are resistors. Not only can it be physically realised electronically, [29] shows that it is possible to represent the learning matrix optically using light. The matrix is trained by applying the input and desired output, then using a binary form of Hebbian learning the weights are determined.

It is possible that pairs of vectors can damage the recall of one another, as 1s are stored in the matrix for one vector they can overwrite a 0 in another vector. This is more likely to occur the higher the number of vectors that are stored within the matrix. Therefore there is some optimal capacity of the matrix for the number of vectors it can store while still performing well at recall. In [29] it is shown that this capacity can be up to 69% of the possible capacity of this size of matrix. They use the method from [69] by Willshaw et al to calculate the capacity. The maximum capacity of the matrix is n^2 where n is the size of the input vector and the maximum capacity of the Learning Matrix is $n^2 ln(2)$ or 69% as much. The maximum capacity is achieved when the matrix is sparsely populated with ones [29].

A strength of the learning matrix is often able to deal with inputs that are corrupted, an example of this is shown in chapter 5 of [54]. In the example an erroneous 1 is set in the input pattern, but the output is still correct. Another strength is the ease of implementation in hardware.

Disadvantages include a restriction to linearly separable problems and the sizes of the input and output patterns needing to be the same.

2.8.3 Correlation Matrix Memories

Correlation Matrix Memories or CMMs are a matrix of integer values used to store data. Each of the inputs and their required output patterns produce a corresponding set of weights. If all of the input and output pairs are considered it is possible to form a matrix of the sum of all of these weights. A correlation matrix memory is an estimate of this matrix using the outer product of all of the pairs of inputs [28].

In order to recall data from the memory an input vector is applied to the CMM. This will provide an output which is composed of the desired output and a ‘noise vector’. This noise vector occurs because of interference between the different relationships that are stored in the memory. It causes the possibility of mistakes when recalling data [28]. It is possible that the input to the CMM is an exact match, a partial match or an overlap [41]. An exact match perfectly maps onto a result stored in the CMM, a partial match has some bits in common with one of the results and an overlap has multiple matches.

A CMM uses a matrix of values to store the information it needs about the classes it will identify. More specifically it uses a binary CMM using only 0 or 1 values.

2.8.3.1 Binary CMMs

Binary correlation matrix memories or binary CMMs have a number of defining characteristics, these are [8];

1. Efficient memory usage; data does not take up much space
2. Training can be done while the system is running, this is advantageous when training a conventional neural network would take too long
3. The possibility to create large processing systems using modular formation of neural networks
4. Easy to implement in hardware
5. Easy to calculate the storage and speed properties of the network

They can be considered to be a matrix of 0s and 1s that indicate if there is a connection present between the input and output at this point. 1 represents a connection. The binary CMM is trained using Hebbian learning; with a pattern being applied to both the rows and the columns of the matrix. Initially all memory locations are set to 0. A connection is formed when both patterns have a 1 present,

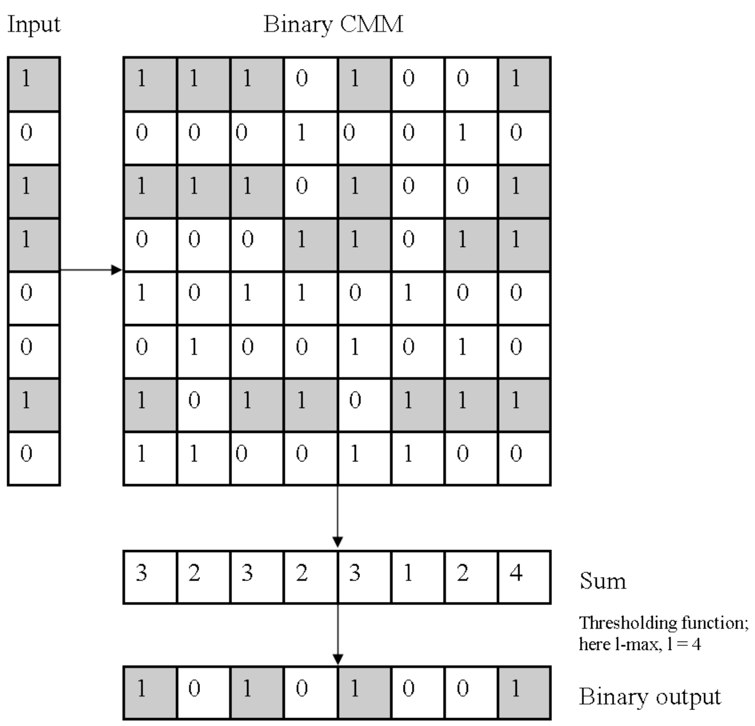


Figure 2.10: An example of recall using a binary CMM.

by placing a 1 in memory. The matrix can then be shown patterns to classify. Recall is performed by applying an input to the matrix, then summing the connections in each column. The result is thresholded in order to find the original pattern. An example of this can be seen in figure 2.10.

Possible choices of threshold function include L-max, Willshaw and Maximum activation. These functions and others are discussed in the following papers [40, 14, 27]. In L-max for example the L highest values are allocated a 1 and all other values a 0. L-max is used in the example in figure 2.10. The choice of the thresholding function allows the system to handle corrupted inputs; for this L-max is preferable to Willshaw. This is because it has more efficient memory usage and it does not suffer from additional bits being set in the output of a perfect match as Willshaw can.

One disadvantage is that small binary CMM systems are limited by poor memory capacity [37]. An advantage is that CMMs use lock-step synchronisation, each output is produced in its entirety and the input and output frequencies are the same. This means that the system does not need to stabilise as a Hopfield network does.

Further disadvantages are the same as for the Learning Matrix. CMMs sizes are also defined by the size of the input and output patterns and are only suitable for linearly separable problems [31].

2.8.3.2 Spiking CMMs

Spiking CMMs are trained in the same way as binary CMMs. They also use the same structure for recall, however the inputs presented to the matrix are spike trains. Each of the matrix elements (neurons) has a state which is determined by the arrival rate of the input spikes and the neuron's previous state [40]. This state can be implemented in the form of the sum of inputs as shown in figure 2.10. When the input spikes occur close together in time there is a large change in the sum, as the spikes arrive further apart from one another the sum only changes by a small amount. The sum can be a leaky integrator, with all of the values gradually decreasing. This means that if no spikes arrive on the input for a certain duration then the sum can be reset back to 0.

A possible disadvantage of spiking CMMs is that their encoding is more complex than a standard CMM. Advantages include a more biological approach and the ability to train the system to produce an output probability rather than a simple 'true' or 'false' response [41]. This method is used by Brewer in [14] for image recognition.

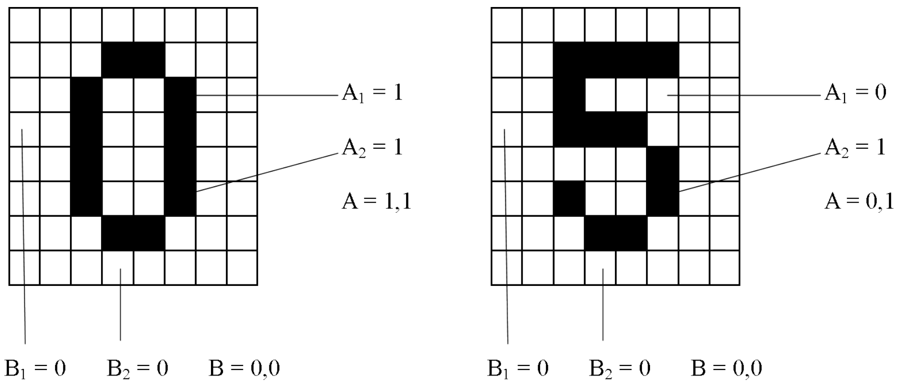
2.8.3.3 Pattern Recognition using CMMs

Pattern recognition is a suitable application for associative memory and it is the method used for testing the Neural Pipeline architecture. For this reason Advanced Uncertain Reasoning Architecture (AURA) [37] is introduced as an example. AURA has been used on applications including 3D face recognition, postal address comparison and trademark matching.

AURA can be considered to be a fully connected one layer neural network that uses binary weights and L-max thresholding [8]. The system uses Hebbian learning for binary data, where a 1 is present in both input and output a 1 is stored in the network. AURA uses an array of CMMs, the training data is split between the CMMs with similar inputs appearing in the same CMM. AURA uses online training which is advantageous because it is easy to add new pattern pairs to the memory [8]. The use of CMMS makes the system easy to implement in hardware [37].

AURA is based on an earlier system ADAM (Advanced Distributed Associative Memory), which was mainly used for image processing. ADAM uses the same method as the Bledsoe and Browning model. In this model a grid of pixels is used for the input, and every pixel is randomly paired up with one other pixel. The values of these two pixels are used to address a location in memory. The system is trained by presenting examples of each of the symbols to be identified to the inputs. For each of the examples a 1 is written into each of the memories at the position which correlates to the symbol being presented, and the value of the two pixels. An example is shown in figure 2.11; memory A has a 1 placed in position ((1,1),(0)) when it is shown the example '0', and another placed in ((0,1),(5)) when the '5' example is presented. The memory is filled up when examples of all ten different symbols are presented. Memory B represents a different pair of pixels, and there are memories that represent all other pixel pairs.

When the training is complete the memories outputs are summed for each of the symbols (here the ten numerals) and the highest one is taken to be the result. Where



Ram A	0	1	2	3	4	5	6	7	8	9
0,0	0					0				
0,1	0					1				
1,0	0					0				
1,1	1					0				

Ram B	0	1	2	3	4	5	6	7	8	9
0,0	1					1				
0,1	0					0				
1,0	0					0				
1,1	0					0				

Figure 2.11: Illustration of how the Bledsoe Browning model stores data based on the example in [54].

there are multiple highest values they are chosen between at random.

ADAM uses a more general version with groups of n pixels, Bledsoe and Browning called these groups 'n tuples'. As n is increased the memory requirements also increase, so there is a physical limit on the value of n. If n is set to 1 then the system is able to recognise presented patterns not identical to its training set, to 'generalise'. However the system is likely to classify incorrectly, because the memory becomes saturated with 1 values. This occurs if there is any difference between training patterns, as 1s are written wherever a black pixel occurs. If n is very large the issue of saturation will not occur however the system is unable to generalise. This shows that there is a trade off between small and large values of n.

AURA differs from this in that it is not confined to 'n tuple' inputs, instead it changes data from numbers or symbols into binary patterns. This allows a larger range of input data than the image based n-tuple ADAM input.

As AURA uses associative memory it can be trained to recognise noisy or incomplete data. One example is that it can be used for boolean sum of products inputs, and is able to identify these even when some of the terms are missing. As inputs with different numbers of terms are stored in different CMMS mistakes can mean that the wrong CMM is accessed, so this must be accounted for. Even if the number of terms is correct the value will be different, so the response threshold must be changed.

2.9 Computer Pipelines

Computer pipelines are an example of an architecture used for keeping a number of processes in time with one another. The aim of the Neural Pipeline is to provide this type timing structure using spiking neurons, so it is beneficial to introduce traditional computer pipelines.

In computing a pipeline is used when a task can be split into several subtasks. A pipeline is advantageous when compared to a single processing unit used to complete the entire task because although the task takes the same amount of time to execute it is possible to operate all subtasks at the same time. This means that when a stream of sequential inputs are provided the throughput is greatly increased.

There are different types of pipeline, the first distinction is the number of different functions that a pipeline can perform. A pipeline with a single function is known as 'unifunctional' and one with more than one function is called 'multifunctional'. The second distinction is whether different function layouts are possible at the same time, if only one is possible the pipeline is 'static' and if it can have multiple configurations at once then it is 'dynamic'. A unifunctional pipeline by definition must be static, but a multifunctional pipeline can be of either type [34].

Pipelines are also classified by the way they perform timing, as synchronous or asynchronous. This is a particularly important distinction considering the aims of this thesis. In a synchronous pipeline each subtask must take the same time to execute and each subtask is able to operate at the same time because of the underlying hardware.

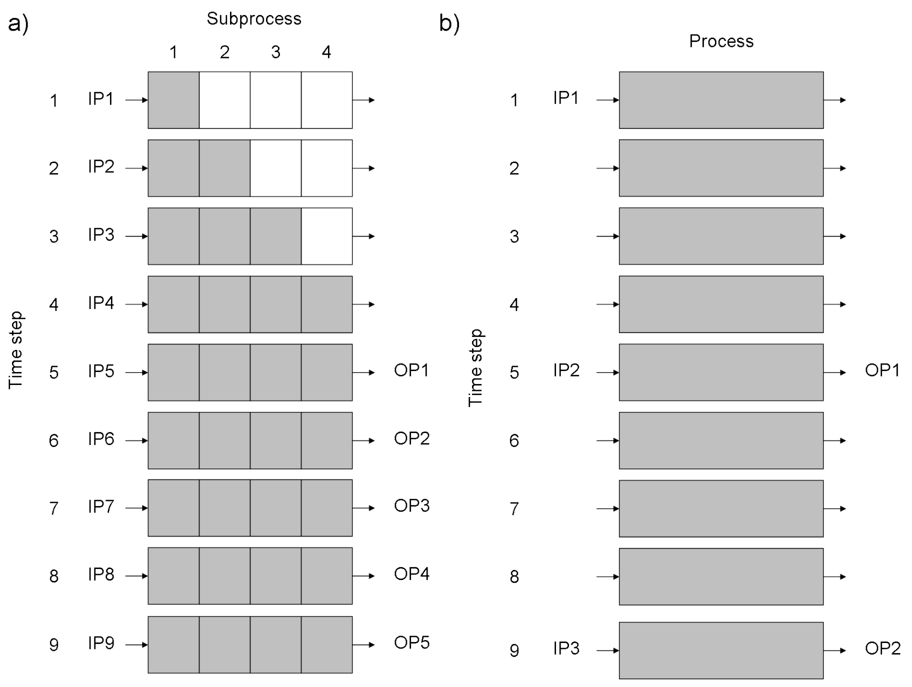


Figure 2.12: A sequence of inputs provided to (a) a pipeline made of 4 10ms subprocesses and (b) a single 40ms process.

2.9.1 Synchronous Pipelines

In a synchronous pipeline all of the subprocesses receive their input at the same time. This process is controlled by a global clock that is fed to each subprocess. This means that the different subprocesses must take the same time to execute.

An example of the advantage gained by using a synchronous pipeline rather than a using single process can be illustrated with a process that takes 40ms that can be split into 4 individual subprocesses. Each subprocess takes 10ms to complete. When a sequential stream of inputs is given to the pipeline each stage will become active in turn and the pipeline will fill up. This process is shown in figure 2.12a from timesteps 1 to 4. At timestep 4 the pipeline has filled up and will return its first output after 40ms, the same as a single process (figure 2.12b). However after this timestep the pipeline can produce outputs every 10ms. If a single process is used rather than a pipeline of 4 subprocesses then it will take 40ms for every input to be processed. This is shown in figure 2.12b.

In a synchronous pipeline architecture a global clock is used to keep all of the processing units in time. The timing is important because each subprocess relies on the previous subprocess for its input, if they get out of time then they will not have the correct input. This can be a disadvantage because it can be a difficult problem to ensure that all of the clock pulses are synchronised. There are other disadvantages including flexibility of the number of outputs processed and different delay lengths for the subprocesses. These are discussed further as advantages of the asynchronous pipeline in the following section.

Synchronous pipelines have the advantage that they use more simple circuitry than asynchronous pipelines. They are also have a predictable output, because they always provide an result a certain number of clock pulses.

2.9.2 Asynchronous Pipelines

Asynchronous pipelines use a handshaking mechanism to control the timing of subprocesses rather than a global clock. The first subprocess sends a request signal, if it receives an acknowledgement signal from the second subprocess then the data is transferred. This means that unlike a synchronous pipeline, where all of the data must be passed forwards at the same time, different subprocesses can pass on data at different times. Their ability to do so is defined by the readiness of the subprocesses before and after. The subprocess before must be ready to provide an input and the subprocess after must be ready to accept the output.

The first asynchronous pipeline was developed by Muller using earlier asynchronous circuit design. Different handshaking mechanisms can be used but the general principle is the same. Each of the subprocesses has a control circuit known as a 'C-element' [61]. The C-element has two inputs; the request signal from the previous subprocess and the acknowledgement signal from the next subprocess. It uses the state of these two signals to provide an output to enable the current subprocess. A change in output state is triggered when both inputs have the same value. The C-elements are chained together so that the activity propagates through the pipeline as the states of the other elements change.

The advantages that asynchronous pipelines have over synchronous ones include the ability to include subprocesses that require different delays in which to process their data. In a synchronous system each of the subprocesses must use the same delay. There is no need to provide a consistent global clock to each subprocess in an asynchronous pipeline. Another advantage is the ability to process a variable number of data items at a given time. If inputs arrive sparsely they will be processed when they arrive rather than waiting for the next clock signal, as with a synchronous pipeline. Lower power consumption is another advantage, because only active subprocesses consume full power. Nowick and Singh [52] provide an overview of all of these advantages.

A disadvantage of asynchronous pipelines is that the hardware is more complicated than for a synchronous pipeline. Another possible disadvantage is that the precise timing of outputs is unknown.

2.10 Synfire chains

The Synfire Chain [3] is an example of a feedforward, biologically plausible, neural architecture. It is the most similar neural architecture to a computer pipeline that has been discovered in the literature.

This architecture consists of a number of layers which are connected using excitatory feedforward connections. Every neuron in one layer is connected to many of the neurons in the next layer. The number of connections from one layer to the next is approximately the same for each layer. The number of neurons per layer is also roughly the same, but the same neuron can appear in more than one layer. The mean number of connections is known as the 'multiplicity' and the mean neurons per layer is known as the Synfire Chain 'width'.

There are two ways in which activity may be passed through a Synfire Chain: synchronously or asynchronously. When firing synchronously the neurons in one layer cause a synchronous burst of firing in the next layer. This synchronous wave propagates through the architecture. When the neurons fire asynchronously there is a gradual build up of activity in the next layer. This increases until the layer is able to stimulate the next layer, and so on through the architecture. In [19] Diesmann et al show that when enough neurons fire synchronously a wave can propagate through the system. With asynchronous behaviour the activity can pass through a number of layers but will disperse. This means that synchronised firing is required to produce a stable response.

In [60] Shinozaki et al explore the use of inhibition to help to enhance the wave of activity as it propagates. They find that inhibition applied prior to the arrival of the synchronised burst of spikes enhances the signal. If the inhibition is applied at the mid point of the burst then it is suppressed. This inhibition is from an external source to one of the layers in the chain. They also test applying excitation and find that it can only enhance the signal and does so when applied during the burst, but it overwrites the timing of the original signal. Therefore an inhibitory signal is found to be more suitable. While this use of inhibition may seem similar to the mechanism used in the Neural Pipeline (see chapter 3) the systems are different. The Neural Pipeline uses inhibition on every layer and the key difference is that the inhibition is triggered by the system itself not at specified times.

A strength of the Synfire Chain is that it is considered to be a biologically plausible architecture. In [19] Diesmann et al consider that the timing requirements they find for synchronous behaviour are consistent with recordings from neurons in the cortex. The architecture is also used for biological modelling. An example of this is the modelling of birdsong [25]. Glaze and Troyer find that a Synfire Chain is able to demonstrate three different properties of birdsong found by analysing samples of real birdsong.

Another strength is that the architecture can also be used for computation, an example of this is image recognition by Arnoldi et al in [6]. The Synfire Chain principle is used to coordinate two separate system modules; one module provides the memory and the other is a detector. Each of these modules contains a Synfire Chain and these two chains are connected, this causes synchronous firing of neurons that represent a particular feature in both modules. The synchronous spiking allows the presented input to be matched to a stored pattern.

A limitation of the Synfire Chain is that it cannot be used on its own to perform a computational memory task. It is a structure that is suitable for timing, but it needs an additional to perform pattern recognition. An example of such a structure is a reservoir computer as introduced in the following section.

2.11 Reservoir Computing

Reservoir Computers are a particular type of neural network composed of two distinct layers, the ‘reservoir’ and a ‘readout’ layer. The reservoir layer is an interconnected layer with connections allocated randomly between its neurons with a certain probability. The network input is provided to this reservoir and is transformed to a particular internal state. The readout layer is connected to the reservoir and these connections are trained to map the input state to the desired output.

The field of reservoir computing includes Liquid State Machines (LSM) [49], Echo State Networks (ESN) and the Backpropagation Decorrelation learning rule (BPD). The differences between the three types are quite subtle; Liquid State Machines tend to use spiking neurons whereas Echo State Networks use analogue ones. Backpropagation Decorrelation trained networks also use analogue neurons but they are distinct from Echo State Networks because there is feedback from the readouts to the reservoir.

An experimental review of all three is given in [66]. The three different methods are not always comparable for each task that they perform. They find that LSM produce fewer errors than ESN for a speech recognition task. This is the case for reservoir sizes of between 50 and 300 neurons. The word error rate is consistently 0.05 less than the best ESN performance. On a different task (Non-Linear Auto-Regressive Moving Average) to model a particular tenth-order system BPD is found to perform comparably to ESN with integrator neurons, but ESN with linear neurons outperform both. Their results suggest that the different types of network are useful for different tasks, and that for recognition tasks the LSM may be advantageous.

Liquid State Machines are the most relevant to this work because they use spiking neurons. The results from [66] also suggest that they may be a good choice for image recognition. An introduction is given below in order to explain their use for training the Neural Pipeline.

2.11.1 Liquid State Machines

Liquid state machines (LSM) were introduced by Maass et al in [49]. LSM are made up of a reservoir of LIF neurons, which is perturbed by inputs. A set of readout neurons are connected to this reservoir to interpret the perturbations and identify what the input was. In [49] they show that LSM have ‘universal computational power’ when used with ideal conditions for continuous inputs, by analysing the system mathematically. This means that the architecture should be applicable to any computational task.

There are two important properties which determine how effective a liquid state machine is. The separation property relates to the liquid layer and the approximation property relates to the readout layer. Separation is the distance between the trajectories of the liquid state when different inputs are presented to the system. The approximation property relates to the ability of the readouts identify the liquid state and transform it to the required output.

The same liquid can be used to solve many different tasks, because only the readout layer is trained. An example of this is given in the experiments in [48]; the different tasks all use the same underlying network. This could be useful in computation because the architecture itself does not need altering to perform different processes. It may also explain how the same brain regions can be used for different tasks.

Liquid state machines have advantages including their biological plausibility and their applicability to different computational tasks. Disadvantages include their high simulation cost and in for this work their lack of a timing structure.

2.11.1.1 LSM for Pattern Recognition

To achieve the aims set out for this thesis the Neural Pipeline architecture has been tested on the computational task of pattern recognition. For this reason the use of LSM for pattern recognition tasks is introduced. As LSM have universal computational power [49] they should be applicable to any computational task. This does not necessarily mean that they are a preferred method for any particular task, however the existing literature shows that LSM are a suitable architecture for pattern recognition.

LSM have been used for pattern recognition for both sound [22, 67] and image recognition [39, 70, 71, 48, 12] as well as producing a simplified model of the visual system [70]. In [48] a LSM is used to recognise an input of a moving object showing that the architecture is suitable for more realistic vision applications.

Kaminski and Wojcik produce and analyse a visual system model made as a multi-layer LSM in their three papers [39, 70, 71]. They use a 16 by 16 pixel grid to provide input to the system, which they break down into smaller blocks (2 by 2, 4 by 4). Black pixels received an input of a randomly allocated train of spikes. They use recognisable shapes as input as well as more abstract patterns. They claim that their model is able to correctly classify inputs even with noise present, but do not provide quantitative results.

Boukhari and Benyettou [12] use a LSM for palm print recognition. They use actual data of 200 individuals from a palm print scanner with a size of 384 by 284 pixels. They compare three learning rules to train the LSM outputs and find a highest accuracy of 98% using backpropagation.

Maass et al [48] use moving image stimuli on an 8 by 8 sensor array sampled every 5ms. Each of the sensors has a value of between 0 and 1. The presented inputs are ‘bars’ or ‘balls’ which move across the sensor grid at different speeds with one of a number of different trajectories. The recognition in this case is not as straightforward as identifying which shape is presented, rather the system was trained to predict the next 25ms or 50ms of activity on the sensor grid. This was achieved with fairly low error rates when considering the task complexity. For the 25ms test there was a 8.5% false positive rate, and a 4.8% true negative rate. The 50 ms case had a 16.5 % false positive rate and a 4.6 % true negative rate. They use unsupervised learning so the system is adaptable to new inputs.

In [22] Fernando and Sojakka achieve sound recognition via image recognition. The image here is of the liquid layer of the LSM, rather than of an input. The liquid layer rather than being simulated as a network of artificial neurons is a physical tank of water. It is perturbed using one or more of four motors, one in each corner of the tank. The water is monitored using a webcam and the image is compressed to a 32 by 24 pixel grid. This grid is used as the input to the readout neurons. This is distinct from the image recognition examples because it is the response from the liquid that is provided in this form rather than the input.

The visual cortex model in [70] and [71] is an example of a multi layered LSM. The layers are mainly connected in a feedforward manner with the only the final layer providing feedback (to the first layer). The input retina is connected to the first layer LGN using excitatory connections. The layers are not examined separately in the papers, they are considered in their entirety. The layers are therefore a topology imposed on the liquid layer rather than functional structures.

Traditional LSMs use a liquid with a randomly chosen connectivity. It is possible to train the liquid to improve the performance for a specific task [51, 33, 15]. This performance difference is measured using the separation property, and an improvement is found using Hebbian learning [51, 15], Particle Swarm Optimisation [33] and multifractal learning [15].

2.11.1.2 Biological Plausibility of LSM

In [49] Maass et al describe the LSM as biologically plausible because it is able to handle time-varying inputs. This is demonstrated in [48] in which Maass et al use a LSM to predict the next position of a moving object. They use the dynamic behaviour to operate and do not need the system to convergence to attractor states to provide an output.

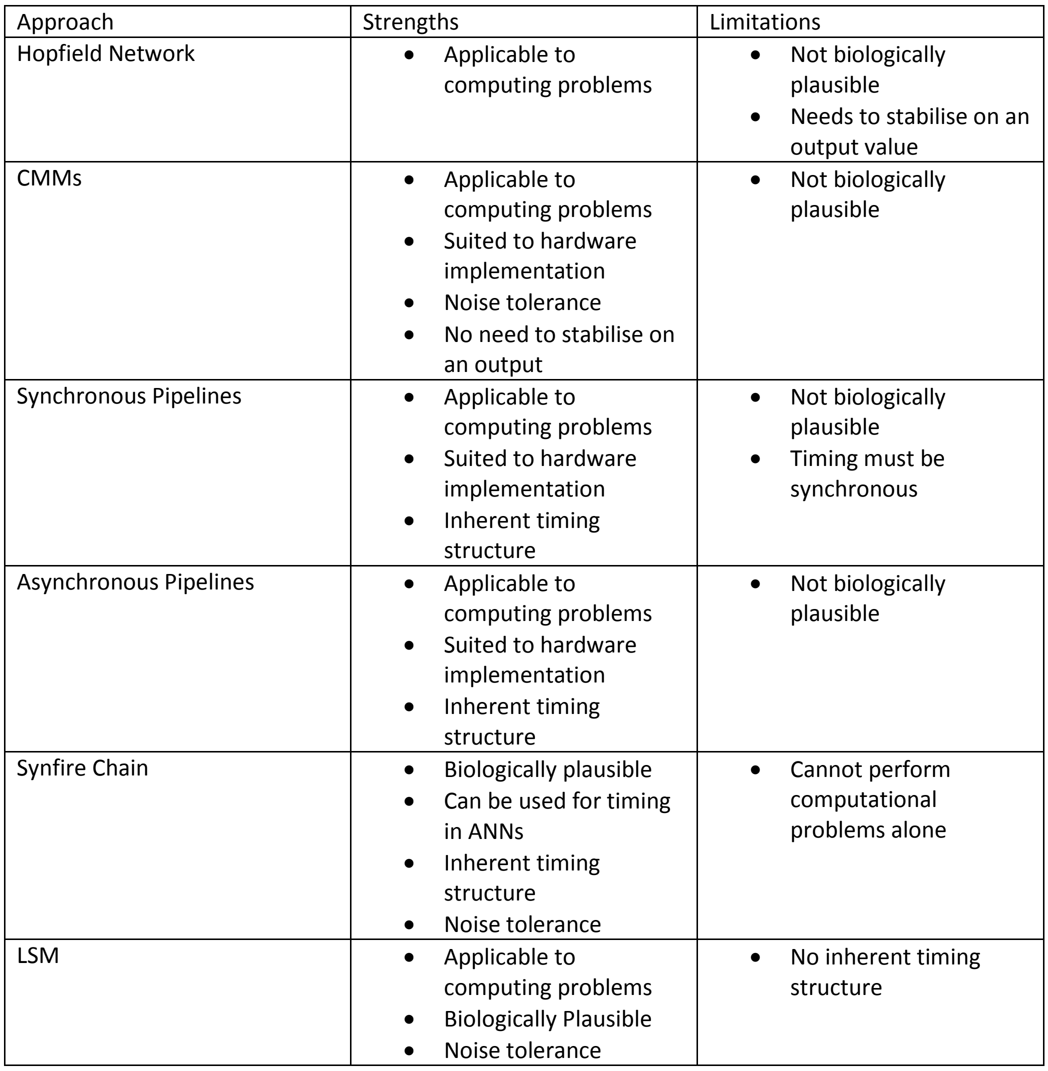
The neuron models and structure of LSM as proposed by Maass et al are more biologically realistic than traditional ANNs. Unlike many artificial neural network models (such as Hopfield networks or Multi-layer Perceptrons) LSM use spiking neurons, these models are more closely matched to real neurons than binary neurons as discussed in section 2.3. In [48] they use models with structures which match microcircuits found from actual recordings of cortical neurons. This means that the structure is biologically plausible.

The connectivity in a LSM can be brain like, but the readout mechanism does not have to be. A computationally suitable readout mechanism can be chosen for testing a LSM but it can be assumed that the brain uses a different mechanism. This split is possible because the reservoir and readout are separate.

The use of LSM for modelling brain areas also suggests that they are biologically plausible architectures. Examples of this are the visual cortex modelled in [71] and the cerebellum in [72].

2.12 Comparison of Architectures

Each of the architectures that have been introduced has different strengths and limitations in relation to the aim of this thesis. A summary of these is given in figure 2.13. The more traditional computer architectures of pipelines and CMMs



Approach	Strengths	Limitations
Hopfield Network	• Applicable to computing problems	• Not biologically plausible • Needs to stabilise on an output value
CMMs	• Applicable to computing problems • Suited to hardware implementation • Noise tolerance • No need to stabilise on an output	• Not biologically plausible
Synchronous Pipelines	• Applicable to computing problems • Suited to hardware implementation • Inherent timing structure	• Not biologically plausible • Timing must be synchronous
Asynchronous Pipelines	• Applicable to computing problems • Suited to hardware implementation • Inherent timing structure	• Not biologically plausible
Synfire Chain	• Biologically plausible • Can be used for timing in ANNs • Inherent timing structure • Noise tolerance	• Cannot perform computational problems alone
LSM	• Applicable to computing problems • Biologically Plausible • Noise tolerance	• No inherent timing structure

Figure 2.13: Comparison of the strengths and limitations of the different architectures for fulfilling the aims of this thesis.

have advantages such as computational efficiency and being well suited to hardware implementation. They are not, however, biologically inspired methods. As the aim in this thesis is to test timing using a neural network architecture the biological plausibility of the method is important. Therefore these architectures do not fulfil this part of the aim. Pipelines are able to fulfil the timing aspect and CMMs the memory aspect of the aim.

The Hopfield network is biologically inspired so fulfils this requirement, but it is not biologically plausible because the outputs have to stabilise. It does not have a timing structure, so does not meet this part of the aim. The biologically inspired methods of the Synfire Chain and LSM have weaknesses such as their additional computational expense. As the work in this thesis is biologically inspired this limitation also applies here. In terms of the aim presented for this thesis the Synfire Chain is limited because it does not have memory, so it cannot be used on its own to perform memory based recognition tasks. LSM do have memory, but they lack the timing structure required by the aim.

None of the architectures presented here provides a neural timing mechanism that can control data within a computer memory. The work in this thesis sets out to demonstrate that an architecture can be developed to achieve the aim by combining desirable features of these architectures.

2.13 Summary

The various fields introduced here build up a picture of ANNs; the biology that they have developed from and how the neuron models have become more complex as computational resources have increased. These models can be built into different structures and can be taught to classify inputs. Different architectures that each partially fulfil the aim have been discussed to provide a basis for the work. The Neural Pipeline architecture introduced in the following chapter to address the aim fully.

Chapter 3

Neural Pipeline Architecture

3.1 Introduction

This chapter outlines the architecture of the Neural Pipeline. It describes the characteristics that are fundamental to the architecture and the other settings that have been used throughout the thesis. The settings can be changed depending on the application, while retaining the structure that defines the Neural Pipeline.

High level design decisions taken in order to produce the architecture are presented in this chapter, with lower level parameter choices described in chapter 4. The preliminary parameter exploration experiments in chapter 4 provide the background required to choose parameters for the experiments presented in chapter 6.

The architecture has been developed so that using the structure of its connections it is able to control how the activity progresses through each of its layers. This has been achieved using the 'external connections'. They are the part of the structure fundamental to the Neural Pipeline architecture. The feedforward connections transmit data between the layers, and the feedback connections control when the layers are allowed to process the data. These inter-layer connections have three parameters; weights, delays and connectivity. The settings that have been chosen for these parameters are given in this chapter.

The layers themselves have a separate set of parameters. There are a number of neurons in each layer with connections between them. These connections have weights, delays and a connectivity value. These internal settings can be changed depending on the task to be completed, unlike the inter-layer connections which define the architecture. The settings presented here are suitable for the tasks that are carried out throughout this thesis.

The three types of behaviour that the architecture can exhibit are defined. The behaviours relate to the ability of the architecture to inhibit each layer in turn after it has had time to become active. 'Correctly inhibited' behaviour has activation for each layer in turn when an input is presented, which is shut off as the next layer becomes active. In the case of 'over inhibited' behaviour at least one of the layers provides more inhibition than is necessary when it becomes active. This stops a second input to the system from reaching the final layer. The definition depends on the required time delay between presenting inputs, if a larger delay is left a system which produces over inhibited behaviour can be made to produce correct behaviour. 'Under inhibited' behaviour occurs when the external inhibition from at least one of the layers is not sufficient to stop the previous layer from spiking.

In summary, this chapter contains the information required to construct a Neural Pipeline. It includes a description of suitable internal settings that can be used,

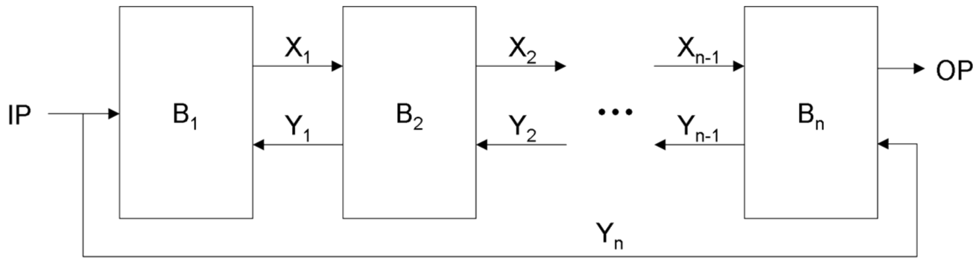


Figure 3.1: The external connections between layers of a Neural Pipeline.

although there are many possible alternatives. The different types of behaviour that the architecture can exhibit are also introduced.

3.2 Neural Pipeline Structure

The Neural Pipeline is composed of multiple layers, there can be as many layers as is required for the task to be performed. Typical sizes presented in this thesis are 3 or 5 layers. Each layer is a subsection of the network, and can be thought of as a network in its own right. The structure that defines the Neural Pipeline are the connections that run between the network layers. Figure 3.1 shows the external connections of the Neural Pipeline architecture. There are feedforward connections which convey the signal from one layer to the next and there are feedback connections that determine when data is allowed to flow. The forward connections are excitatory to allow the signal to pass through the system. The feedback connections are inhibitory and are used to stop a layer from receiving input while it is processing. If a layer is active then the previous layer, which provides it with an input, is inhibited. If a layer is inactive then the previous layer is uninhibited and can provide an input. It is these inhibitory connections that provide the sought-after means of the architecture controlling its own coordination using the data in the system (introduced as the aim in section 1.2). Figure 3.2 shows how the activity flows through the system using these external connections.

The inhibitory connections must come from within the system, rather than being applied from an external source (as Shinozaki et al do in their paper [60]). This is because having an external source means that the timing of data flow is controlled externally, rather than by the system itself. This does not solve the issue of timing, just moves it outside the system, much like a clock. It would be an option if the timing were triggered by the next layer becoming active, but this step just adds extra overhead. Equally it is not suitable for a layer simply to inhibit itself after a delay, because each layer should be free to operate until the next layer is sufficiently active. This means that a fixed delay may prematurely silence the layer.

The inputs to the system are presented to a subset of the first layer, this subset are known as 'input neurons'. The input can be of different sizes depending on the application. An example input size used in this thesis is 81 neurons out of 100 in the first layer. This is used for all of the learning experiments that are presented in chapter 6.

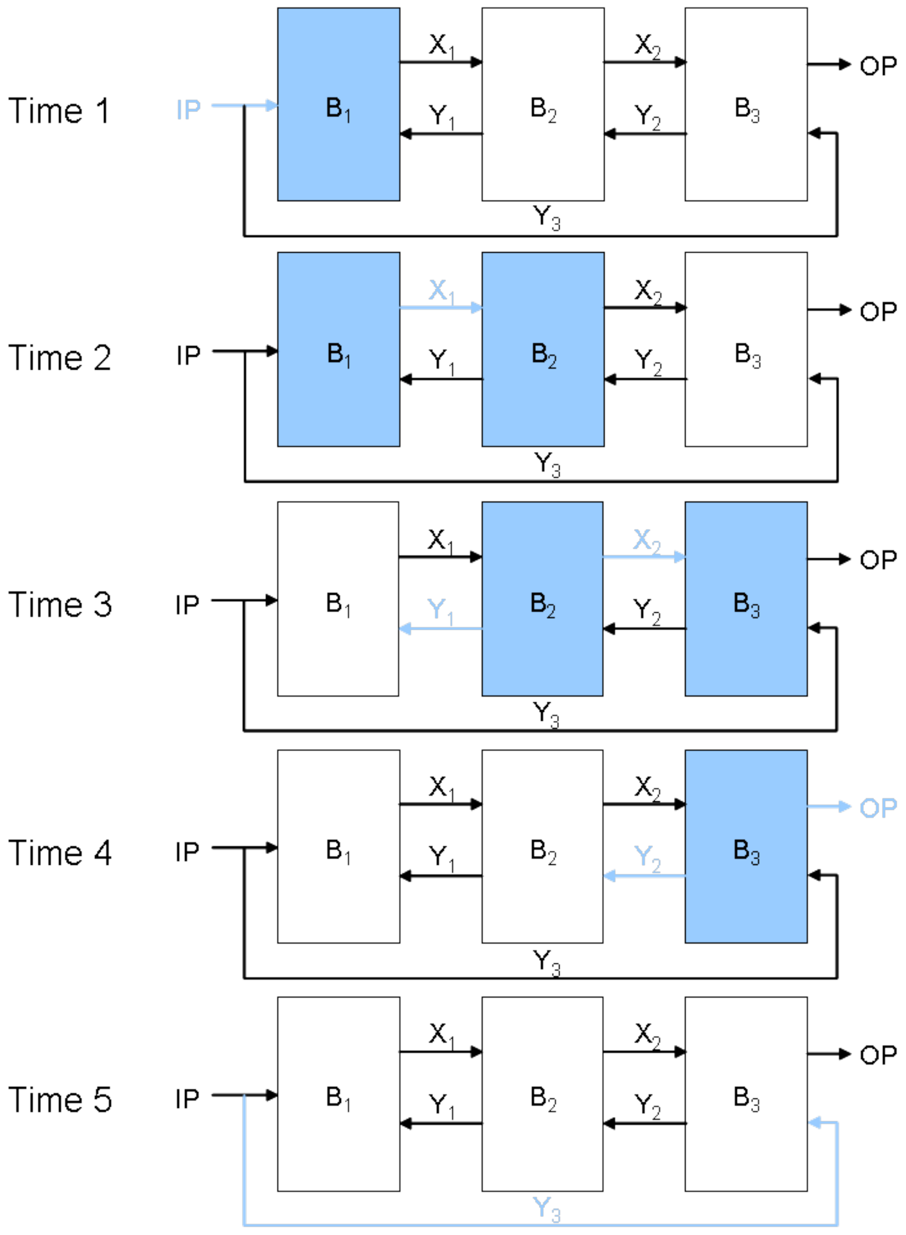


Figure 3.2: The activity flow through the system when one input is presented. The grey shading represents where the activity is at that time.

The feedforward connections between layers are labelled X; in figure 3.1. These connections have an excitatory weight of 5 and a delay of 1ms. The external excitatory connections run from every neuron in layer n provide input to a randomly chosen subset of neurons from layer *n+1*. The subset for each neuron is chosen independently, with a uniform chance of picking any of the neurons in layer *n+1*. The connectivity used for the Neural Pipeline is 1/10 of the layer size.

The feedback connections between layers are labelled Y; in figure 3.1. These connections are inhibitory with weight -0.3 and are used to suppress activity in layer *n-1* when layer n becomes active. None of the Y connections have excitatory values because they would just introduce noise into the previous layer, and encourage spiking rather than suppressing the activity. There is a time delay of 5ms on these inhibitory connections to allow the layers some time to operate before they are shut off. The inhibitory connections run from all neurons in layer n to all neurons in layer *n-1*.

The last layer of the architecture is inhibited in a different way to the other layers. This is because it has no 'next layer' to provide it with inhibition. This layer must be inhibited because it has the same internal settings as the other layers (see the following section for details of these internal settings), so relies on inhibition to stop the layer spiking. If the layer is not inhibited then it will continue to spike and no further inputs will be able to pass through the system.

The inhibition to the final layer is provided from the system input. This is shown as connection Y_n in figure 3.1. There is a delay of 10ms multiplied by the number of layers on the connection. The delay is chosen to be long enough to allow the last layer to become active but not so long that it continues to spike for much longer than the other layers.

The inhibition to the last layer could be provided from any point in the system. It is chosen to be from the input so that there is flexible behaviour depending on the length of the delay on the connection. If the delay on connection Y_n (in figure 3.1) is longer than the time taken for the activity to reach B_n, then the stimulus will suppress its future self. If the delay is shorter than the time for activity to reach B_n then the last layer (B_n) is permitted to remain active until another input is presented to the system.

It can be seen that the structure of the Neural Pipeline (figure 3.1) closely matches that of an asynchronous pipeline (see [52] figure 1 for a simple diagram). The feedforward connections are the equivalent of the asynchronous pipeline's request signal and the feedback connections the equivalent of the acknowledgement signal. The overall activity flow in figure 3.2 is also comparable to that of an asynchronous pipeline, with the activity propagating through the layers sequentially. The main distinction is that the neural pipeline has a more fluid transition between states. The timing diagram is a simplified representation, showing the layers as either on or off, but in reality the layers gradually transition as the neurons stop firing. Another difference is that the feedforward and feedback connections are not digital signals (as the req, ack signals are), but a series of connections that propagate spikes.

3.3 Layer Structure

Each layer in the architecture is effectively an individual network that is able to perform a task. The external connections do not interfere with the internal functionality of the layer, so any type of network could be used as a layer. The external connections transfer data between the layers and shut them off when no input is to be provided to the next layer.

The type of layer that has been used throughout the experiments presented here is a randomly connected group of neurons. Each neuron has a set number of connections and each connection target is chosen uniformly from all of the neurons in the layer. This randomised connectivity is useful for two reasons. When examining the behaviour of the system the random connections can be considered to be a network that has been trained to do something unknown. This means that the behaviour of the system can be examined for different examples of network without training. The second reason is that, when used for learning (in chapter 6), the randomly connected network can be used as a liquid state machine.

For practicality of simulation time, sizes of 100 neurons or fewer have been used for the majority of experiments presented in this thesis. These sizes have been shown to be sufficient for the learning tasks carried out in chapter 6, and should extend to larger examples.

The neurons in each layer are all leaky integrate and fire neurons, with the parameters given in the following section. The neurons are either excitatory or inhibitory to follow Dale's Principle (see section 2.2.3). This means that all of the outgoing connections from the neuron have the same sign, positive for excitatory or negative for inhibitory. All neurons can receive connections of either type. Each layer has an equal number of excitatory and inhibitory neurons.

The connections within the layer are known as the 'internal connections'. All of the neurons in a layer have the same connectivity. That is to say, the same number of connections from them to neurons within the layer. The neurons and their connections are described in the following sections.

3.3.1 Neuron Type

There are many different types of neuron model that are used for artificial neural networks. An introduction to the different types was given in section 2.4. They vary from the simple binary ones to more complex spiking types. Spiking neurons were chosen to use for the Neural Pipeline architecture, because they are more biologically realistic and are able to represent more complex behaviour.

The specific type of spiking neuron that has been chosen for the architecture is the leaky integrate and fire (LIF) neuron (introduced in section 2.4.1). LIF neurons were chosen for several reasons: they represent a simple model of spiking neurons and therefore strike a balance between computational efficiency and biological realism. They are also used for a Liquid State Machine in [49] and for analysis of a randomly connected network in [56], allowing for easier comparison with these results. Only LIF neurons have been tested in this thesis, but the architecture could use any type of neuron. This is a possibility discussed in the further work section 3.5.1.

The simulations presented in the thesis use the NEST neural simulator. The architecture is not restricted to a particular simulator or even to a computer simulation. The simulation represents the easiest method of identifying how successful the architecture is under different conditions and suitable parameter choices. It is possible that the architecture could be constructed in electronic hardware or even using actual neurons. This is another possibility presented in the further work section.

The neuron parameters that have been used for the tests using the architecture are given in table 3.1. The neuron parameter names in the table relate to the ones used in NEST, but the same values could be used in different simulators. All parameters with the exception of the threshold are set to the defaults for the NEST simulator. This decision was made because the parameters are biologically sensible and because it is necessary to have a starting point from which to adjust the other system parameters, these settings would appear to be a good starting point. Static synapses are used for all experiments.

The threshold value is set to be lower than the more biologically realistic value of 50mV, which is the default simulator value. This is because the size of the network is very small compared to that of a biological neural network. Here there are typically a few hundred neurons, as opposed to the approximately 10 billion neurons in a human brain [28]. In order for the neurons to spike with only a limited number of inputs the threshold value was reduced. The value of -69.931mV was chosen by running an experiment, with three layers of 100 neurons and an input spike train of 10ms (as is used in most of the experiments through the thesis). The threshold value was reduced until this spike train produced enough spikes in layer 1 to propagate to layer 2.

The LIF neuron is modelled in NEST using equation 3.1. The variables are set to the values given in table 3.1. The neuron input I_{syn} is generated by spikes that appear on the input of the neuron over time. The spikes are produced by other, connected, neurons in the network. Each individual spike produces an alpha current, the shape of which is calculated using the alpha function given in equation 3.3. An example of the function shape is graphed in figure 3.3. I_{syn} is calculated by summing these inputs over a given time as shown in equation 3.2. In this equation I_p represents a list of all input spike times.

These equations are taken from the NEST documentation, which can be accessed by downloading NEST [24].

$$\frac{dV_m}{dt} = -\frac{(V_m - E_L)}{tau_m + I_{syn}(t)} \times \frac{C_m}{(C_m + I_e)} \tag{3.1}$$

$$I_{syn}(t) = \sum_{t_j \in I_p} [w_j alpha(t - t_j)] \tag{3.2}$$

$$alpha(t) = e \times (t/tau_s) \times (e^{-t/tau_s}) \times StepFunction(t) \tag{3.3}$$

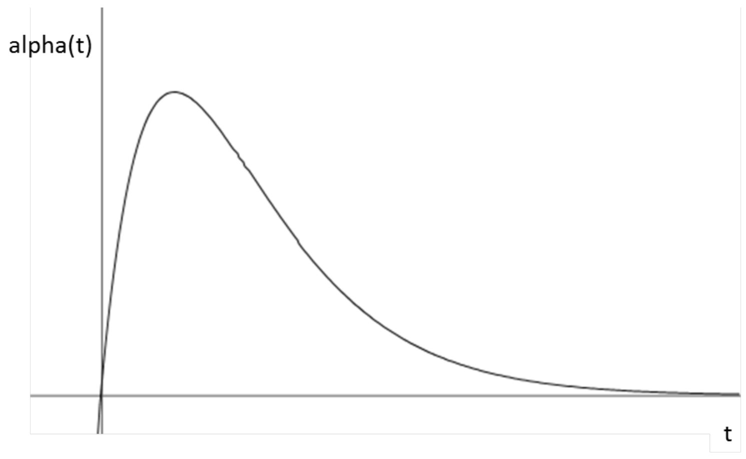


Figure 3.3: The graph shape of the alpha function used in the NEST simulator as given in equation 3.3.

Neuron Model		
Leaky IAF neuron		
Variable name	Value	Description
V_{th}	-69.931	Threshold voltage in mV
E_L	-70.0	The resting potential of the membrane in mV
C_m	250.0	Membrane capacitance in pF
tau_m	10.0	The time constant of the membrane in ms
t_{ref}	2.0	Length of the refractory period in ms
V_{reset}	-70.0	The reset voltage in mV
tau_s	2.0	Synaptic alpha function rise time in ms

Table 3.1: Neuron parameters.

Model Summary	
Internal excitation	0.5
Internal inhibition	-0.5
External excitation	5.0
External inhibition	-0.3
Internal delay	1.0 ms
External excitatory delay	1.0 ms
External inhibitory delay	5.0 ms
External excitatory connectivity	10%

Table 3.2: Connection parameters.

3.3.2 Connections

Each of the neurons within a layer is given the same connectivity value. The connectivity is the number of connections from the neuron to other neurons within the layer. A connectivity of 10 is found to be the most versatile for providing correct behaviour when other parameters are varied. The 10 connections from each neuron to neurons in its own layer are chosen at random. There is an equal probability of choosing any of the neurons within the layer as the connection target, including the neuron itself and neurons that it has already been connected to. This allows for multiple connections between a pair of neurons and also allows for self connections.

The weights on these connections are all set to *+w* for excitatory connections and *-w* for inhibitory ones. A variation is to use weight values randomly chosen from a range *0 to +w* for excitatory connections and *0 to -w* for inhibitory connections. The value of *w* is one of the parameters used to control the level of activity in the system. All internal delays are set to 1ms, this is because the connections have to have a delay that is not 0 (a simulation constraint). 1ms was chosen as a short delay. The same value is used for all the connections because this allows the delays on the external connections to control the data flow, without delayed signals from inside the layer altering the behaviour.

Table 3.2 gives typical values for the internal parameters, chosen using the work in chapter 4. These are used for the learning experiments that are presented in chapter 6.

3.4 Fundamental types of Behaviour

The Neural Pipeline architecture is a system that has been designed to handle a stream of inputs. When the architecture is presented with two or more inputs in sequence, there are three types of behaviour that it can exhibit.

These three behaviour types are labelled ‘under inhibited’, ‘correctly inhibited’ and ‘over inhibited’. They have definitions based on how the activity flows through the system. Correctly inhibited is, unsurprisingly, the desired type of behaviour. Under and over inhibited are undesirable behaviour types, with under inhibited considered to be worse than over inhibited. The reasons for this are described below.

Examples of these three types of behaviour are illustrated in figure 3.4 when two inputs are presented 30ms apart. All neurons are initialised to be silent, having experienced no prior activity, and there is no background noise.

Correctly inhibited behaviour has the following definition:

> 'Upon the presentation of an input each layer becomes active in turn and after a time is suppressed'

In all of the cases presented in this thesis the system has no background noise, so the suppression should make the layer silent for a time. This is not a requirement for correct behaviour, the suppression must just be to the 'normal' background level.

An example of correct behaviour is shown in figure 3.4 (a). In this case activity from both inputs can be seen in each of the layers, and importantly the activity is suppressed again after activation.

Over inhibited behaviour is defined as:

> 'For any of the inputs the activation of layers in turn stops before the last layer'

'Over inhibited' behaviour is shown in figure 3.4 (b). In this case the inhibition from layer 2 is too strong, because the second input does not produce any activity in the first layer and therefore any subsequent layers.

Over inhibited is the preferable of the two undesired types of behaviour, because it is easier for the system to recover from. This behaviour is dependent on the required time between inputs, because the inhibition in the layer slowly returns to the resting potential, over time. This means that instead of presenting the second stimulus after 30ms as shown in figure 3.4 it could be presented after 50ms. By 50ms the inhibition within the first layer would have decreased and the input would be able to make the layer active and pass through the system correctly.

Under inhibited behaviour has this definition:

> 'Any of the layers resumes spiking after suppression before another input arrives at the layer'

The least desirable type of behaviour is shown in figure 3.4 (c), this is known as 'under inhibited'. When there is too little inhibition between layers *n-1* and *n*, layer *n* fails to suppress the activity in *n-1*. This means that layer *n-1* continues to fire and prevents any other activity from being provided as input. If this occurs then the pipeline needs to be flushed of activity before any further inputs can be provided.

Infrequently, when compared to the other responses, a run which appears to be a combination of under and over inhibited is produced. An example is shown in figure 3.5.

These responses are classified as under inhibited, because 'under inhibited' is the over-riding behaviour, as it is the least desirable type of behaviour. Even if

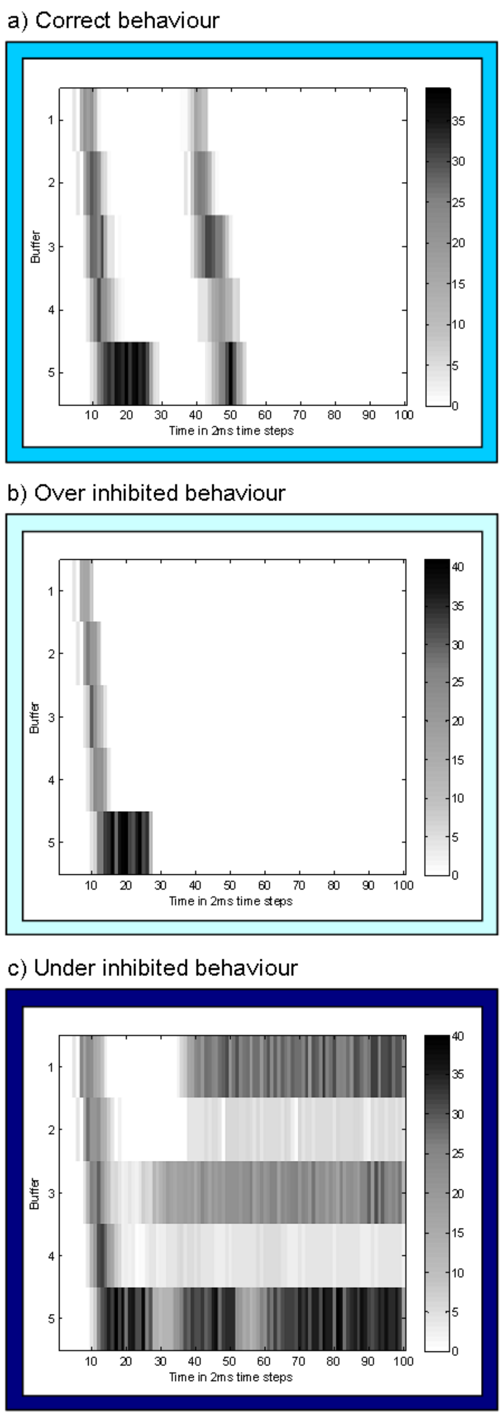


Figure 3.4: Examples of the three types of behaviour that can be exhibited by the Neural Pipeline.

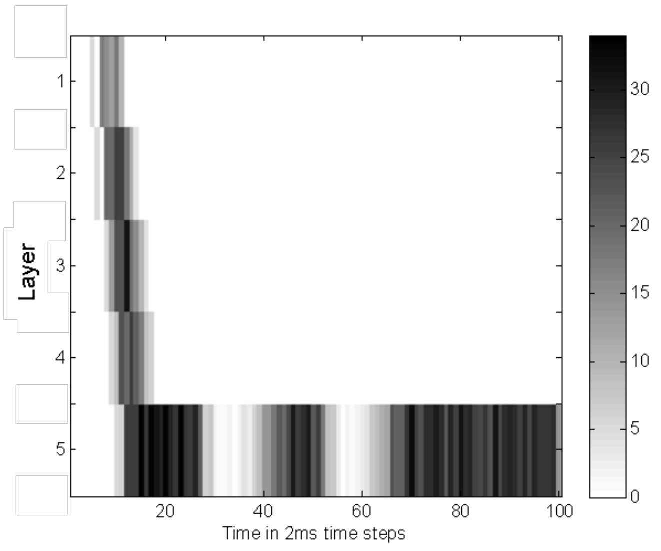


Figure 3.5: An example of a run that exhibits both under and over inhibited behaviour. The second input does not appear in layer 1, which is a symptom of over inhibited behaviour. At the same time layer 5 continues to spike after it has been inhibited, which indicates under inhibited behaviour.

the over inhibited behaviour was overcome by waiting longer before applying the next input, the under inhibited behaviour would still be present. That is the reason for this classification. Only correct behaviour is required for the system to work, so although this property is interesting it has not been investigated further. It is therefore recommended for investigation in the further work section 3.5.1.

The correct behaviour seen in figure 3.4 is the same behaviour that would be seen in an asynchronous pipeline. The activity propagates through each layer in turn as the layers communicate using feedforward and feedback connections. The other two types of behaviour are not seen in asynchronous pipelines.

3.5 Discussion

The architecture has been developed as a neural network that can control the timing of data flowing through the system using its own firing neurons, rather than an external source. The fundamental structure of the pipeline that allows this is the external connections. These connections are the part of the architecture that defines it as a Neural Pipeline. The layers are connected using excitatory forward connections that pass the data from one layer to the next. The feedback connections between the layers are inhibitory and they are used to shut off the layers. The last layer also requires inhibition and this is provided from the system input.

The weights and delays on the external connections can be changed depending on the application. The connectivity could also be changed, but the choice of connectivity is more constrained. For example, to preserve the signal information between one layer and the next the excitatory connections cannot run from all neurons in one layer to only one neuron in the next.

The internal parameters of number of neurons, weights, delays and connectivity are necessary in any artificial neural network. It is important to note that the parameter choices are used to make the Neural Pipeline operate correctly for the tasks presented here but they are not inherent to the architecture. Any artificial neural network could be used as a Neural Pipeline layer. The weight on the external inhibitory connections should be adjusted depending on the level of activity that the layer provides. It is possible for different layers to have different settings, because the inhibition for each layer can be adjusted separately.

There are three types of behaviour that a Neural Pipeline can exhibit. They relate to the balance between activity in the layers and the amount of inhibition on the external connections. The required behaviour is 'correctly inhibited' and this occurs when the inhibition is balanced with the activity. This causes all of the layers to become active and then be shut off in turn, and for the data to pass through the entire architecture. There are two undesirable types of behaviour: 'under inhibited' and 'over inhibited'. Over inhibited behaviour occurs when the inhibition between the layers is large compared to the activity within them. It inhibits a layer so that it is unable to respond to the next input, a solution to this behaviour is to increase the time between inputs. Under inhibited behaviour is caused when the external inhibition is low compared to the activity within the layer. It means that the data in a layer is not suppressed sufficiently by the inhibition, so the first of the inputs continues to spike on one of the layers. This prevents new inputs from passing through the system.

3.5.1 Future Work

During the development of the Neural Pipeline architecture, decisions have had to be taken about which way to progress. Due to the scope of the thesis not all avenues could be explored. The decisions taken were determined to be the best way of progressing the architecture and the reasons that they were chosen are described in the relevant sections. This section describes the alternative proposals, they are suggested as further work.

The layers in the versions of the Neural Pipeline architecture tested here have all used leaky integrate and fire neurons. Different types of neurons have not been tested. There are alternative spiking models and non-spiking neurons as discussed in section 2.4. It is believed that the architecture will work independent of the type of neuron that is used. The architecture has not been developed specifically for the LIF neuron, this has just been the mechanism for testing it. Future work includes testing the system with different neuron types to make sure that this is the case.

Once the architecture has been tested further it is possible that it could be implemented in hardware. The simulation is easier to test and make changes to, so the production of hardware is suitable for a more developed version. The main advantage to having a hardware implementation would be the speed that the system could run. The easiest method of producing a hardware version would be to use some existing hardware such as an FPGA. The hardware implementation on FPGAs may be able to use CMMs to represent the layers. CMMs are suitable for implementation on FPGAs [42]. The weight values on the connections within the layers would be represented as the values in the CMM matrix. There is also the possibility of testing the architecture using real neurons, to test the biological plausibility.

The mixed behaviour mentioned in section 3.4 represents another possibility for investigation. It is believed that this behaviour is caused when one or more of the layers exhibits under inhibited behaviour and one or more of the layers exhibits over inhibited behaviour. It is also thought that the over inhibited layer (or layers) must appear before the under inhibited layer (or layers), because otherwise the behaviour would only be under inhibited. Understanding what causes this mixed behaviour may make it easier to balance the behaviour across the layers. It should help to confirm whether it is necessary for all of the layers to behave correctly independently, or if it is possible to balance under and over inhibited layers to achieve correct behaviour overall.

A possibility for further analysis of the architecture is to compare the dynamics with the Synfire Chain (see section 2.10). In the Synfire Chain only synchronised firing of neurons allows stable propagation of activity through each layer. It is believed that the internal connections and the feedback in the Neural Pipeline mean that synchronised activity should not be required for a stable system. This is because the layers can stimulate themselves to produce a lot of activity and then be made quiet by feedback. In the Synfire Chain the layers do not stimulate themselves so synchronised spikes are needed to stop the activity dispersing. The ability of the architecture to perform without the neurons being synchronised is hinted at by the lack of a synchronous firing pattern in the layers when inputs are presented. Further work is required to demonstrate this.

Dale's principle has been used within the layers, but has not been extended to the external connections. It is possible to use inhibitory inter-neurons to handle the external inhibitory connections from excitatory neurons. Implementing this should result in exactly the same behaviour as the architecture currently exhibits, because it is an implementation detail. To make the feedforward connections follow Dale's principle too, one possibility is to connect only the excitatory neurons to the next layer. This may impact the behaviour, because less of the data will be transferred between the layers. It is believed that the reduction of activity may improve the probability of producing a correct run because it will reduce the activity in the later layers. It is also thought that the information will be reduced, because only half of the neurons are able to communicate with the next layer. This will make the identification of distinct inputs more difficult. A second possibility for the excitatory connections to follow Dale's principle is to add a circuit of neurons that will allow an inhibitory signal to be converted to an excitatory one. This option would add computational expense to the network, because an additional two neurons are required for each of the connections, plus connections to a new spike generator. All of these additional spikes must be processed, so this mechanism is not very efficient. The two different options should be tested and compared to the existing system (without Dale's principle on the external connections). This will show whether removing the external connections from inhibitory neurons removes information and therefore whether the overheads of the second option would be preferable. These suggestions are explained further in appendix A.1.

Another possibility for future work is to consider alternative architectures based on the Neural Pipeline. It is thought that a variation using two Neural Pipelines, one for control and one to process data, could provide additional functionality but still operate in the same way. It would allow one signal to provide an input and a different signal to control the timing. This is a more versatile architecture but testing would be required to show that it could perform in the same manner as the existing architecture. It is discussed further in appendix A.2.

A useful way of extending the work would be to introduce generalisation, so that the system is able to classify similar inputs or noisy inputs as the correct output. In the examples in chapter 6 there is no generalisation. A suggested way of achieving this comes from the way that different inputs follow different trajectories through the state space, that is to say that different neurons fire at different times. If similar inputs take similar trajectories then it may represent a way of generalising. This is discussed further in appendix A.3. Investigating the trajectories of different inputs is therefore a useful area for future work.

3.5.2 Summary

The Neural Pipeline is a multi-layered computational architecture that uses a combination of excitatory and inhibitory connections to control information flow through the system. The layers can theoretically contain any neural network, here they are composed of LIF neurons with randomised connections. The architecture demonstrates one of three types of behaviours; correctly inhibited, over inhibited or under inhibited. The following chapter introduces the background work used to make the decisions for the parameters to use in the architecture in this chapter.

CHAPTER 4

Preliminary Parameter Exploration

4.1 Introduction

The Neural Pipeline has been designed as a neural network architecture that can control its own data coordination. The structure that achieves this is a series of neural network layers, with two sets of connections running between them. One set carries information forwards to be processed in the next layer, and the other set running in the opposite direction suppresses the activity in previous layer. This suppression stops input arriving at a layer for a time, allowing it to process the current signal.

Initial parameter investigations are presented in this chapter, along with the decisions that were made based on their results. As the simulations here were part of the development of the architecture they use different parameter settings to each other. Different parameters are introduced at some stages, they are based on the results found here. The points at which they were introduced and why are described in this chapter.

There are three types of behaviour that the Neural Pipeline can exhibit (correct, under inhibited and over inhibited), that are independent of the task that the architecture is carrying out. Investigations into how the behaviour changes as parameters are varied is introduced.

These investigations include testing the parameters to achieve correct behaviour, and investigating the transitions between types of behaviour. The parameter tests for the external and internal connections are performed on the weights, delays and connectivity. The requirement for inhibition to the last layer was discovered during development. The reasons for the introduction of this inhibition are discussed. The settings within the layers themselves are investigated. They include weights, delays, connectivity and number of neurons. Changes to the size of the system input are also discussed.

All simulations of the Neural Pipeline (at the various stages of development) are carried out using NEST, the NEural Simulation Tool [24]. The reasons for choosing this environment are outlined in section 4.2.

The parameters must be investigated to identify suitable values to use in computational tasks. In this work there are no on-line updates to the parameters while the system is running, they are set to particular values for each experiment. As this work is developing the Neural Pipeline as a new architecture it is important to investigate the parameters manually to gain an understanding of their influence.

Although the experiments are separated out into their relevant sections, many examine two or more parameters. The tests show that to achieve correct operation there are no ‘perfect’ values, rather a balance between different parameters must be achieved. In particular the balance between the activation within a layer and the external inhibition is found to be important. This is described in more detail in the discussion section of the chapter (section 4.9).

4.2 Simulation Environment

For efficiency an existing simulation environment was used for the simulations, rather than implementing such an environment from scratch. Another advantage of the available environments is that they include implementations of many different types of neuron, so extensions to the architecture do not require that these new types are implemented.

Many standard packages used to simulate artificial neural networks for Computer Science (such as Matlab’s Neural Network Toolbox) do not include spiking neuron models. As an initial decision for the Neural Pipeline was to use spiking neurons (as introduced in section 3.3.1) these types of environment were not suitable.

Other environments are specifically designed for computational neuroscience, so include spiking neurons. Examples of these environments include Neuron [30], NEST [24] and BRIAN [26]. The simulators provide similar functionality and although there may be advantages and disadvantages to each of them it is not possible to review them without simulating the network in each of them. The investigation in this case is not to review simulation environments, but to use one to investigate the Neural Pipeline architecture. An extensive review of types of simulator can be found in [13]. The simulation environment that has been used throughout the investigations into the Neural Pipeline is NEST (NEural Simulation Tool) [24]. PyNEST is a Python interface for NEST, this was chosen as the implementation method because it is simpler to use than the alternative SLI language [21].

4.3 Interpreting the Behaviour

Figure 4.1 shows how the columns in the behaviour graphs presented in many sections of this chapter are made up. Each simulation run produces an output which can be graphed in the same way as the behaviour graphs in figure 3.4. From this trace each individual run is classified according to their behaviour type. The column shows the total number of each type of behaviour.

The behaviour of the system is investigated in the following sections. These sections explain how the different parameters of the architecture have been tested to identify values which promote correct behaviour.

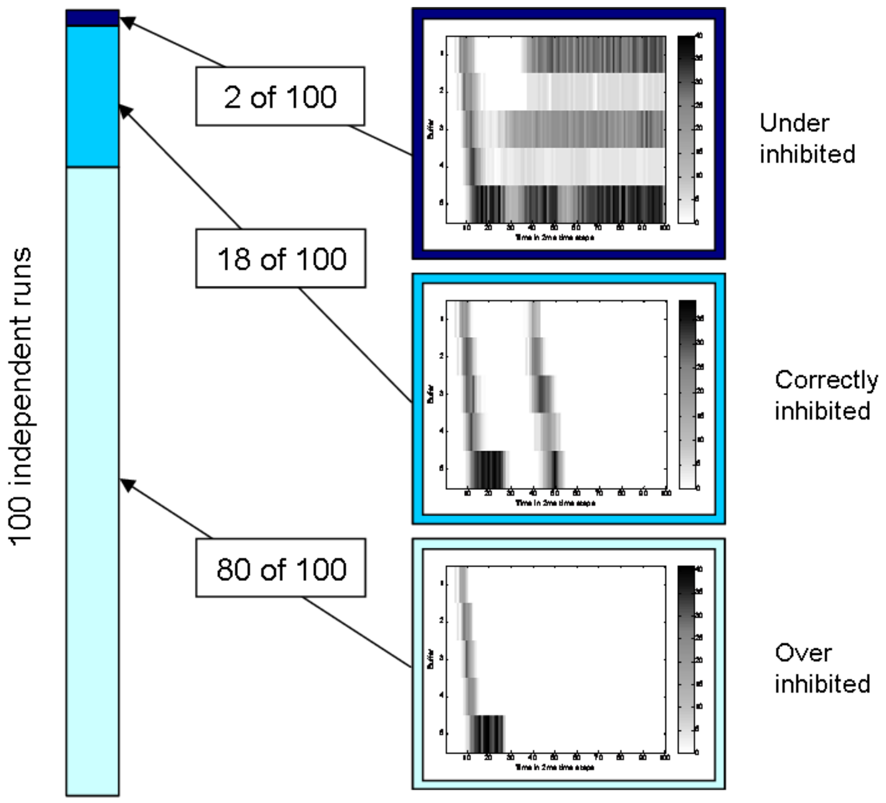


Figure 4.1: This diagram explains how the graphs of behaviour should be interpreted. This bar is an example from figure 4.12 (a) which contains all three types of behaviour with the same parameter settings. The bar represents 100 simulation runs, all with the same parameter settings but with different randomly chosen connections within the layers. Each of the three colours represents a behaviour type. In this case 80 of the 100 runs are over inhibited, so have a trace similar to that shown for over. 18 of the runs are correct and 2 are under inhibited.

4.4 External Excitation

The external excitation provides the input from one layer to the next (these connections are labelled X in figure 3.1). The weights on the connections dictate how strong the input to the next layer will be. The delays determine how long the input takes to arrive after the previous layer is stimulated. The connectivity itself determines which inputs can be distinguished, and how many of the neurons in the layer receive input from the previous layer. All of these choices have an impact on how the system behaves.

4.4.1 Weights

To identify the influence of the external weight values on the architecture different values were tested, while leaving the other parameters fixed. The tested network has 3 layers, each with 5 neurons with a connectivity of 3. The behaviour of neuron 1 is representative of the entire layer because in this experiment the external excitatory connections run from all neurons in layer n to just neuron 1 in layer $n+1$.

Examples of the tests are shown in figure 4.2. All of the graphs in (a) show the membrane potential of the first neuron in layer 2 and the graphs in (b) show the membrane potential of the first neuron in layer 3. The graphs show how the change in excitatory weight alters the membrane potential when the same input is provided to the system (graph (i) has the lowest value of excitatory weight and graph (iv) has the highest). From the graphs it is possible to see that as the excitatory weight is increased that the B2 neuron goes from producing some spikes (graph i) to being very strongly inhibited (graph iv), represented by the large drop in membrane potential. The oscillations seen in graphs (ii) and (iii) are seen in the transition to under inhibited behaviour. The layer is inhibited (shown by the troughs in the graph) but is able to recover and spike again (the peaks in the graph). At the same time as the B2 neuron is strongly inhibited, the corresponding trace for the B3 neuron (figure 4.2(b)(iv)) is spiking constantly. This is typical of the under inhibited behaviour, with one layer continuously spiking and inhibiting the previous layer.

The important aspect of the figure is the graph shape, illustrating whether the neurons are firing or not. The actual values of membrane potential do not give any further insight, because they were performed on an early version of the Neural Pipeline, so are not representative of the current architecture. **The key point is that if the external excitatory weights are set to be high, when compared to the threshold of the neurons and the magnitude of the input, then the behaviour can change from correct to under inhibited.** Different parameter setups of the Neural Pipeline will have different values where the behaviour switch occurs. This is a future avenue to be explored (see section 4.9.1)

4.4.2 Delays

To allow the layers time to process the current data, there needs to be a delay on either the external excitatory or external inhibitory connections, or on both sets. It is the sum of the delay on the excitatory and inhibitory

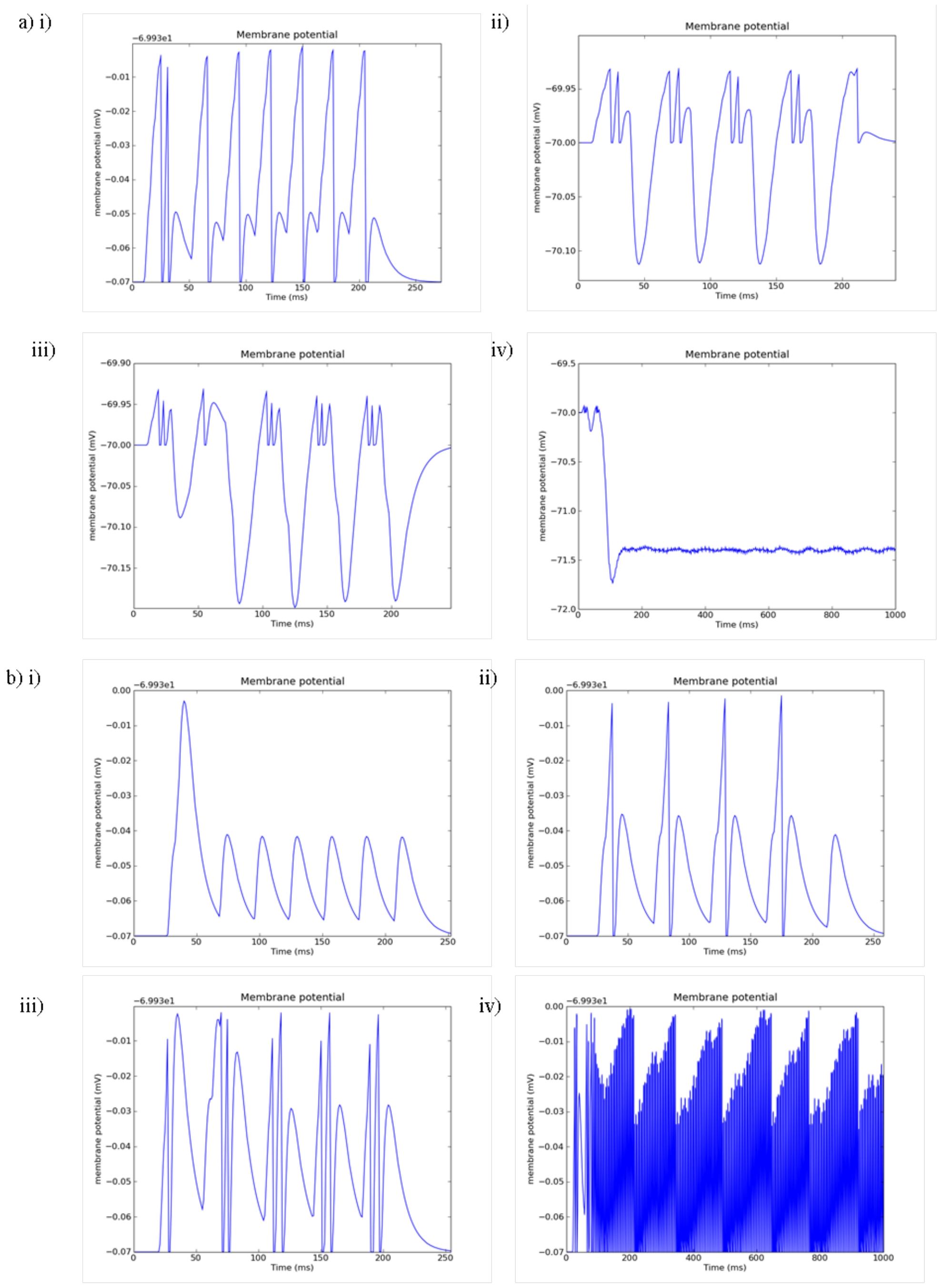


Figure 4.2: Each subplot shows the trace of membrane potential over time for a particular neuron. The (a) plots show neuron 1 in layer 2 and the (b) plots show neuron 1 in layer 3. This neuron is representative of the layer activity because the layers are fully connected to neuron 1. The excitatory weight value is increased from plots (i) through to (iv). (i) has value 2.0, (ii) 2.125, (iii) 2.875 and (iv) 3.0. The peaks in the plots represent spike events. Comparing plots (i) and (iv) shows that as the weight is increased the behaviour changes from correct to under inhibited. Correct behaviour can be seen by the low level spiking in both (i) plots compared to the suppression seen in (a)(iv) and high levels of excitation in (b)(iv). These together are characteristics of under inhibited behaviour.

delays that dictates the time available for the layer to process data. For all other experiments on the Neural Pipeline architecture the delay has been set on the inhibitory connections, so that the forward activity is transferred quickly between layers, but there is a delay before the previous layer is shut off. This decision is arbitrary because the total time that the layer has to process the data is the same either way round.

For the tasks presented in this thesis the absolute time that the architecture takes is not considered important. It is the processing of data in a relative order as it progresses through the layers of the architecture. For this reason the choice of delay in milliseconds is not examined. A total of 6ms is chosen as a suitable delay, because it gives each layer time to become active before being shut off. In real world problems the absolute time will be important, so absolute time is a consideration for future work.

To test that the behaviour is the same whether the larger delay appears on the excitatory or the inhibitory connections, a simulation was run with the delay of 5ms on the excitatory connections and 1ms the inhibitory ones. The result of presenting a 'square' twice to such a system can be seen in figure 4.3 (b) with results of presenting same input to the original system (5ms delay on the inhibitory connections) shown in (a). There are subtle differences in the two responses, but they are very similar. The most noticeable difference is that the activity in layers 2 and 3 takes longer to start, resulting in a graph which looks stepped. This is an expected consequence, because the delay means that the activity takes longer to be passed to the next layer. There is also a little less spiking in layer 3 graph (b) this is easily explained. It occurs because the activity takes longer to reach layer 3 when the delay is on the excitatory connections, but the delay to the final layer is the same in both cases. This means that the last layer has less time to spike in graph (b) than (a).

The length of the delay on the inhibitory connections has been fixed at 5ms with the excitatory delay fixed at 1ms throughout the tests on the Neural Pipeline architecture. This amount of delay allows correct behaviour to be seen with many different parameter settings (all correct runs presented here use this delay) and is sufficient to allow recognition of the inputs presented in chapter 6. As it has been usable for all of the experiments presented here, the delay length has not been examined. This is a task suggested for future work (see section 4.9.1). The reason to investigate the parameter is so that a suitable value can be chosen for different requirements. It should become increasingly important as the tasks that the architecture is being trained to carry out become harder, because the time required per layer is likely to increase.

Another aspect to be investigated is the influence of delay on over inhibited behaviour. The length of time between inputs has some control over whether a system is correct or over inhibited (described in section 3.4). The delay also defines how many inputs can go through the system within a particular time, this suggests that the type of behaviour will have a relationship with the delay length. Another possibility to consider is both excitatory and inhibitory connections having a delay, though as with switching the delays this should perform in the same way as the original system, just with a greater delay.

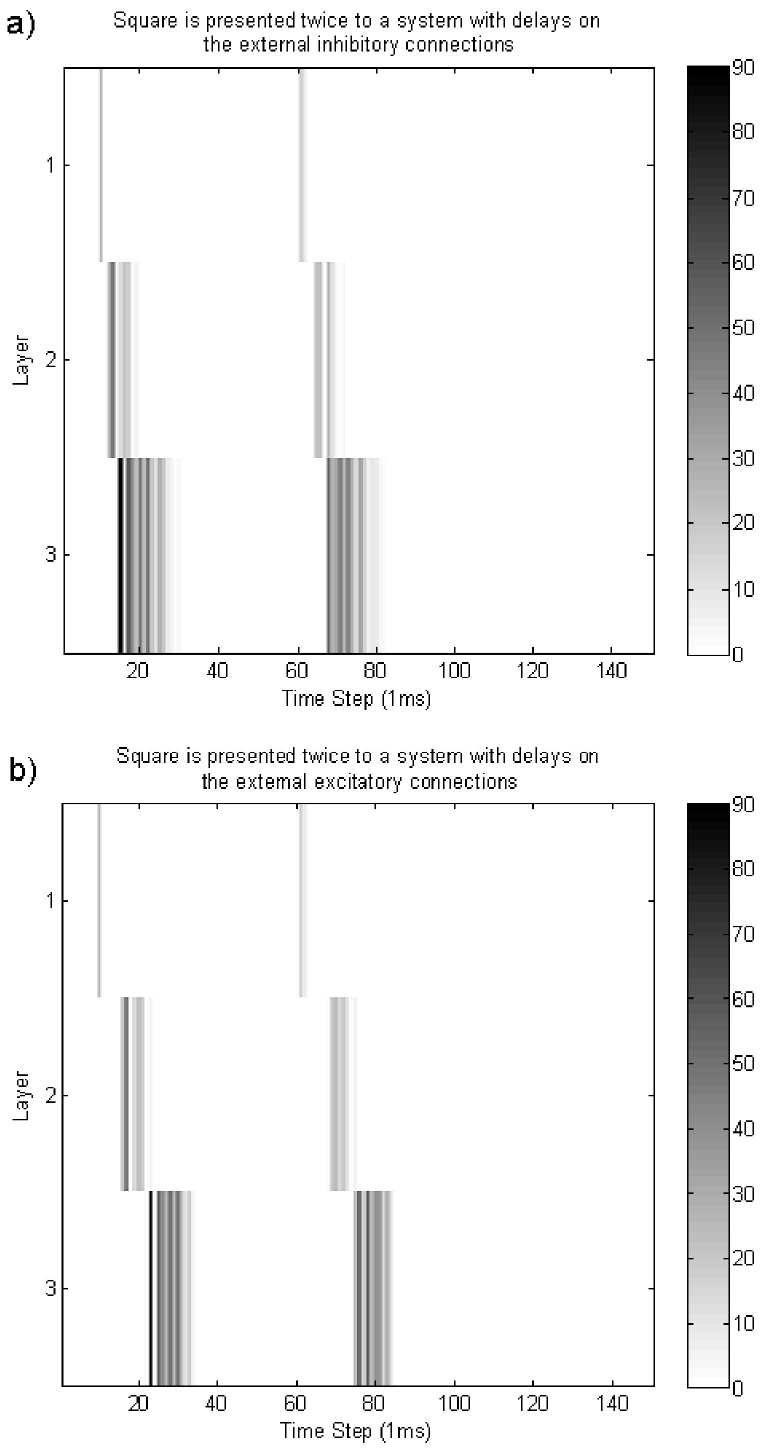


Figure 4.3: The result when square is presented twice, at times 1ms and 50ms to a system with (a) 1ms delay on the external excitatory connections and 5ms on the external inhibitory connections and (b) 5ms delay on the external excitatory connections and 1ms on the external inhibitory connections.

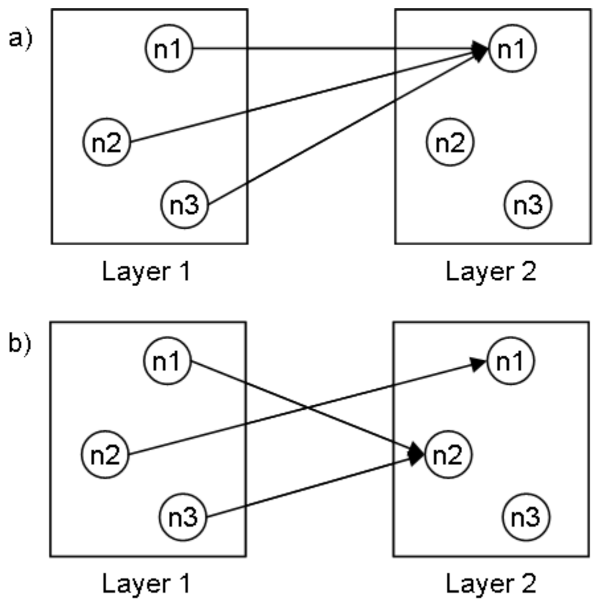


Figure 4.4: The different external excitatory connectivity patterns that have been tested (a) all neurons in layer 1 are connected only to neuron 1 in layer 2 and (b) neurons in layer 2 are connected randomly to a subset of the neurons in layer 2, the subset here is of size 1.

4.4.3 Connectivity and Topology

The majority of the tests in this chapter were carried out using excitatory connections to only the first neuron in the layer, as shown in figure 4.4 (a). This is acceptable when carrying out tests on the behaviour without learning, but once learning is introduced to the system this type of connectivity is no longer appropriate. This is because all of the activity is filtered through the first neuron, making each input look the same in the second layer.

To allow the different inputs to retain their differences from the second layer onwards an alternative connection pattern was chosen. It connects each neuron to multiple neurons in the next layer and therefore allows the different inputs to retain their differences from the second layer onwards. This alternative is to connect each neuron in layer *n-1* to a different, randomly chosen, subset of neurons in layer *n*. An example of this, with a subset of 1 neuron, is shown in figure 4.4 (b). The setting is 1/10 of the number of neurons in the layer, so that it scales with the layer size. Thus a value of 10 connections per neuron was used for all experiments with 100 neurons per layer. This was found to provide enough activity in the layer, without over exciting it. It is because of the possibility of over excitement that a fully connected version of excitation has not been used. Another reason for choosing to use a sparse connectivity over a fully connected set is that each neuron in one layer connects to different neurons in the following layer. This is likely to improve the separation of different inputs as they progress through the system, so this is the preferred choice.

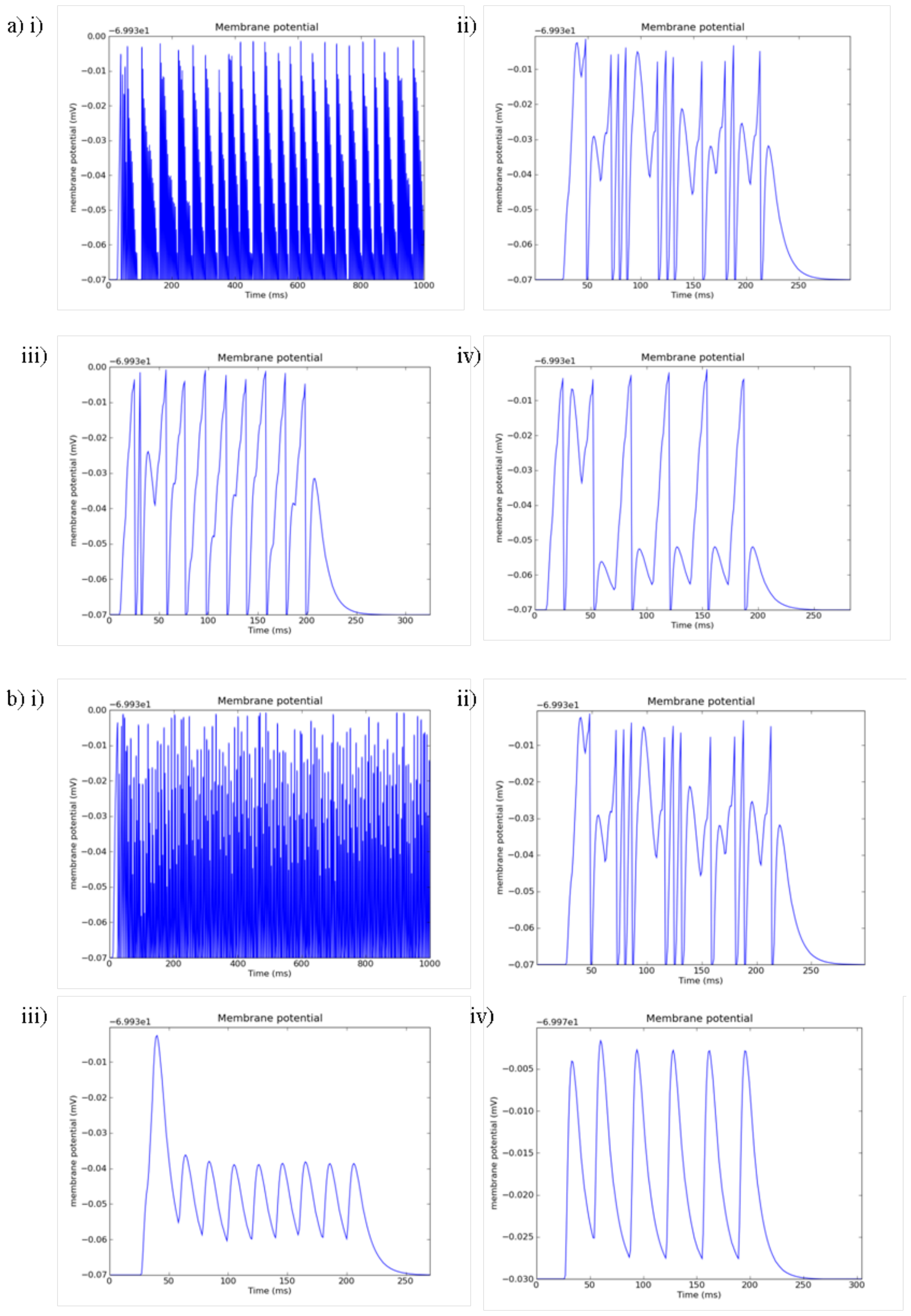


Figure 4.5: Each subplot shows the trace of membrane potential over time for a particular neuron. The (a) plots show neuron 1 in layer 2 and the (b) plots show neuron 1 in layer 3. This neuron is representative of the layer activity because the layers are fully connected to neuron 1. The inhibitory weight value is increased from plots (i) through to (iv). (i) has value 0, (ii) 1, (iii) 5 and (iv) 20. The peaks in the plots represent spike events. Comparing the plots from (i) through to (iv) shows that as the weight is increased the behaviour changes from under inhibited (with no inhibition) to over inhibited. When under inhibited the traces show large amounts of activity. When over inhibited there is barely any activity in the traces.

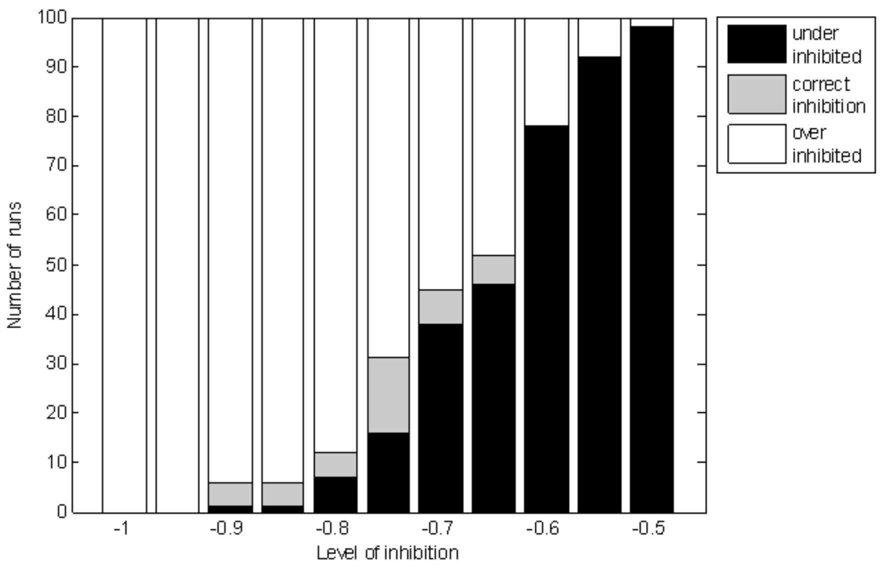


Figure 4.6: The type of behaviour exhibited by the Neural Pipeline with different external inhibitory weight values. 100 runs of each value were carried out and the behaviour of each of the runs coloured by type.

4.5 External Inhibition

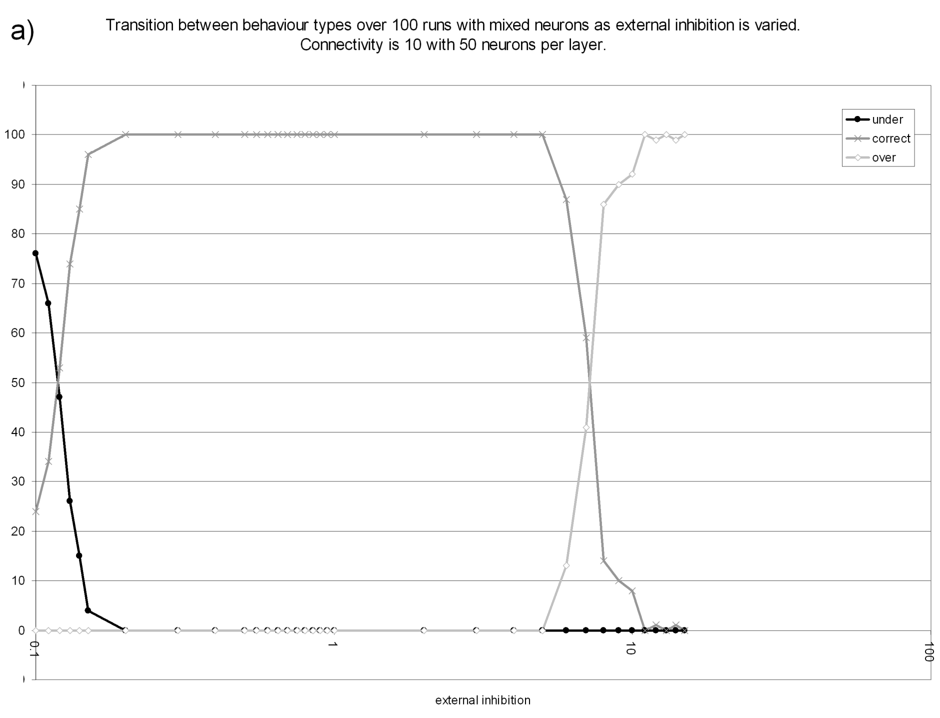
The external inhibitory connections provide feedback from one layer to the previous layer (they are represented as the Y connections in figure 3.1). As with the excitatory connections the weights, delays and connectivity must be decided upon. Additionally for the inhibition there is the consideration of how to inhibit the last layer, because it does not have a 'next layer' to inhibit it.

4.5.1 Weights

The external inhibitory weights have an impact on behaviour. Initial tests demonstrate that **without any inhibition between stages, the resulting behaviour is, unsurprisingly, under inhibited.** This can be seen in figure 4.5 graphs (a)(i) and (b)(i) when compared to (a)(ii) and (b)(ii). Both (i) graphs show that the system continues to spike until the end of the simulation at 1000ms, but both (ii) graphs show that the inhibition suppresses the activity after spiking. The input is presented for 200ms and with inhibition spiking finishes soon after the stimulus stops.

The graphs also show that the larger the level of inhibition, the fewer spikes there are. This can be seen by the reduction in changes of membrane potential from graphs (i) to (iv). The inhibition will start to take effect after 6ms due to then total delay on the external connections, so it is active for the majority of these graphs.

Although it can be seen that (i) is under inhibited when compared to (ii), because it continues to spike until the end of the simulation, it is not possible to say from these results whether the behaviour (in graphs (ii) to (iv)) is correct or over inhib-



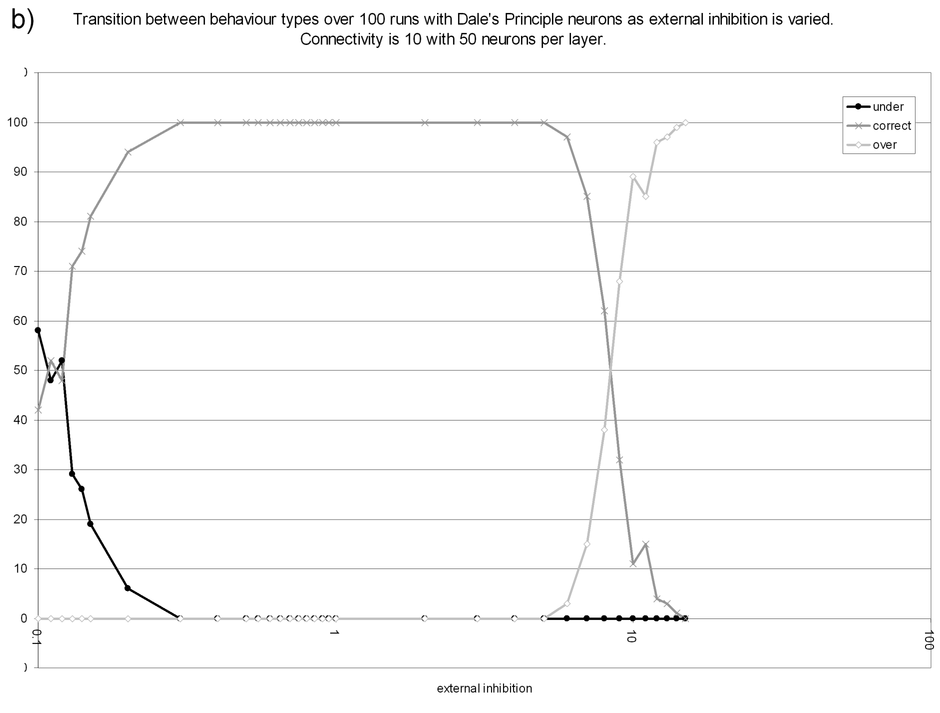


Figure 4.7: The number of runs with each type of behaviour, with 100 runs for each of the inhibitory values plotted. The graphs show the transition between the different types of behaviour. Graph (a) shows the result when using mixed neurons and graph (b) shows the result using neurons that follow Dale's Principle.

ited. This is because only one input is presented to the system, so it is not possible to see whether the second input is prevented by the inhibition. The influence of the inhibitory weight on the system behaviour has been investigated further by comparing 100 runs for inhibitory weight values between -0.5 and -1. The results of this are shown in figure 4.6. They show that there is an optimal value of inhibitory weight of -0.75 for these parameter settings. In the graph even with the optimal value of -0.75 for inhibition only 15% of the runs had correct behaviour, this is not a reflection of a poor choice of inhibitory weight. Instead it is the high level of excitation, caused by the high connectivity value, that is overriding the behaviour. No choice of inhibition is able to compensate for this. Once the excitation is reduced the system can achieve 100 correct runs using an external inhibitory value of -0.75 (and a range of other values). This is shown in figure 4.7.

Figure 4.7 shows tests on a wider range of inhibitory weight value than figure 4.6, with values between -0.1 and -15. The figure compares mixed neurons in graph (a) with Dale's principle neurons in graph (b). These graphs show that with the correct value of connectivity (here it is set to 10) it is possible to achieve 100 runs all with correct behaviour for a wide range of inhibitory values. The internal weight values have are +2.5 for excitatory connections and -2.5 for inhibitory ones, and the graphs show that the system can have 100 correct runs with external inhibition in the range 0.3 to 5. There are some correct runs within the range 0.1 to 14 with both types of neuron. Both types of neuron have almost the same response to the different inhibitory weight values, with the one main difference being that the drop off of correct behaviour is slightly steeper using mixed neurons. This is shown by the less rounded appearance of the 'correct' line on the mixed neuron graph in figure 4.7 (a).

Examples of the types of response that the system gives with different values of inhibitory weight are shown in figure 4.8 (b). Each one of these graphs represents one individual run from the graph shown in figure 4.6. When compared with the behaviour definitions (section 3.4) they demonstrate that **high external inhibition results in over inhibited behaviour, and low external inhibition results in under inhibited behaviour.** These types of traces are used to determine the behaviour type for all of the graphs of behaviour.

4.5.2 Delays

The delay on the external inhibitory connections is set at 5ms for all of the experiments using the Neural Pipeline, with the exception of the test described above in section 4.4.2. This delay has been sufficient for all of the tests performed here, but it is likely that it will need adjustment depending on the particular task. Investigating this is an extension to the work outlined in section 4.9.1.

4.5.3 Connectivity

The connectivity of the external inhibition influences the behaviour of the system. Figure 4.8 (a) shows example runs of different numbers of connections. With those parameter settings the fully connected system, 1/2 connected and 1/3 connected are

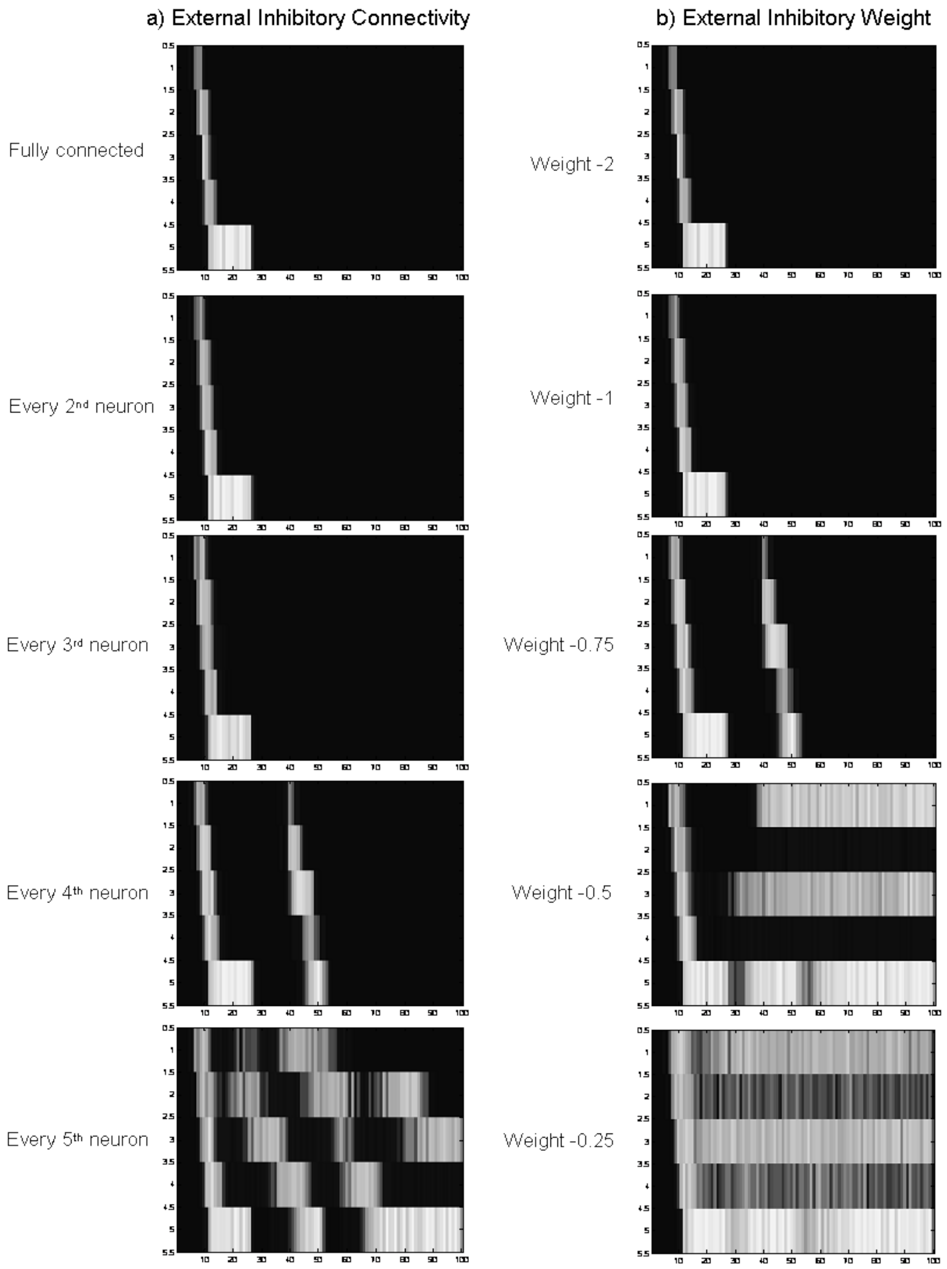


Figure 4.8: A visual representation of examples of different types of behaviour with (a) different external inhibitory connectivity and (b) different external inhibitory weights.

all over inhibited. 1/4 is correctly inhibited and 1/5 is under inhibited. The result seen with different levels of connectivity depends on the other system parameters. This is not shown by these graphs, but by the other graphs presented in this chapter because all of the other experiments use fully connected external inhibition. So for example later in figure 4.10 all correct runs are achieved with full connectivity on the external inhibition. As the example in 4.8 (a) is over inhibited, this shows that an adjustment of other parameters can have an impact.

The decision was taken to fully connect the external inhibition from each layer to the previous one in the Neural Pipeline architecture. Although from the graphs presented in 4.8 (a) this may not seem an intuitive decision, as shown through this chapter, many different parameters influence the correct behaviour of the system. It is therefore sensible to fix at least one of the parameters and to balance the other ones to achieve correct behaviour. It was determined that the connectivity would be fixed at 100% but using a lower weight than the examples presented in figure 4.8 (a).

This was decided because for correct behaviour it is necessary to reduce the spiking in a layer to the usual background level. No noise is used in these tests, so this background level is silent. This complete suppression is not a biological constraint, but is a decision taken for ease of identifying the behaviour types. The higher the background level of spiking, the more difficult to identify from the plots whether the behaviour is correct or under inhibited. The quickest method of stopping all neurons at the same time is to have inhibitory connections to all of them.

4.5.4 Inhibition to the Last Layer

The last layer in the system (layer B_n in figure 3.1) has a set of inhibitory connections running from the system input. These connections were introduced into the architecture because without giving the last layer different parameters to the other layers (which would also be possible) **it relies on the inhibition to shut it off after finishing processing.** Without inhibition the final layer continues to spike until the end of the simulation as shown in figure 4.9 (a). This is under inhibited behaviour, for the obvious reason that the last layer is not inhibited. Figure 4.9 (b) shows the associated problem that any further input cannot get through because layer 4 is being heavily inhibited by the continuous spiking of layer 5. The same simulations with inhibition to the last layer are shown in 4.9 graphs (c) and (d). The final layer is now stopped after a set delay. Graph (d) shows that two inputs presented to the system now result in correct behaviour. Both inputs go through all of the layers and are shut off after a time to process.

The delay set on the inhibitory connections can either be longer than the time it takes data to traverse the entire system, or shorter. When it is set to be longer the input will stop its own activity in the future. This could be achieved by attaching the inhibitory connection to any of the layers in the Neural Pipeline. The reason that the inhibition is connected to the input, rather than to the final stage, is so that the delay may be set to be shorter than the duration of the whole system. This means that rather than stopping itself, an input will stop the signal that is currently active

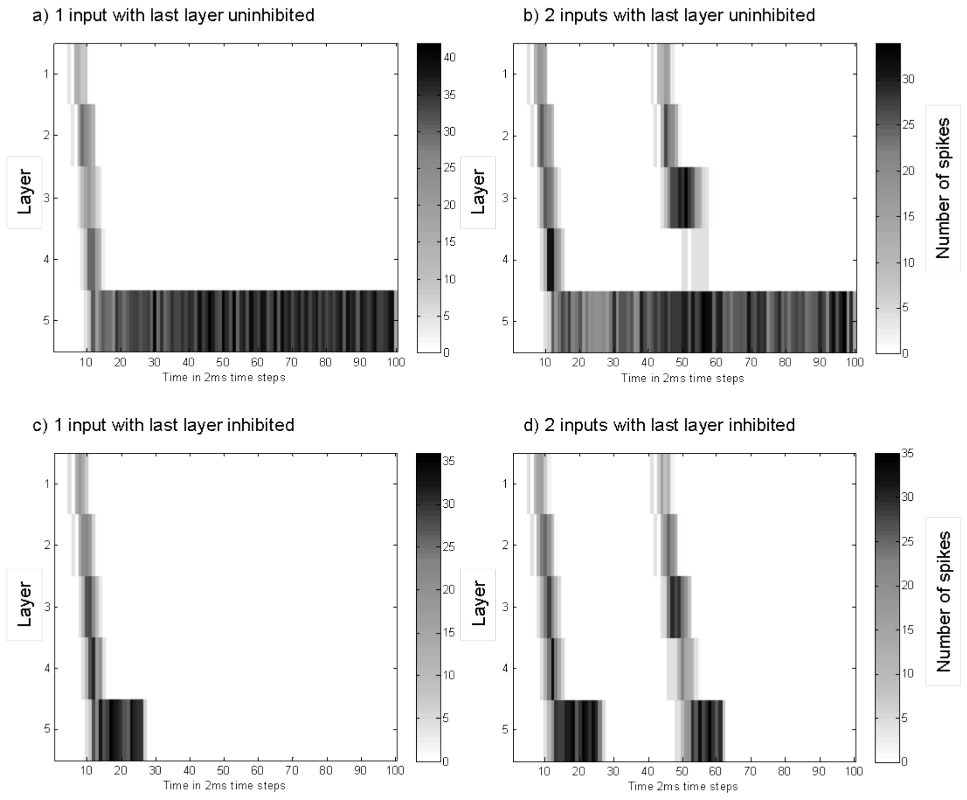


Figure 4.9: Graphs to show a particular instance of the consequences of having no inhibitory connection to the last layer (a and b) and having an inhibitory connection (c and d).

on the output. This means that the output can be left on for as long as necessary before the next signal comes.

For all of the experiments presented here the delay has been set to be longer than the time for data to go through the Neural Pipeline. The use of a shorter delay is an avenue for future work (section 4.9.1). Even with this constraint the actual length of the delay in ms needs to be considered. The length of delay needed depends on how the system is set up; with higher numbers of layers the delay must be longer.

The parameter choices for these connections are the same as those for the external inhibition; weight, delay and connectivity. These values have been matched to the ones used for the external inhibition connections by factoring in the difference in the input magnitude and timing. For this reason the delay has been fixed at 10ms multiplied by the number of layers. 10ms was chosen as in the initial tests it takes less than 5ms for the activity to pass through each layer to the next. 10ms is used as an overestimate because the delay can be too long and the system will still operate correctly, it just means that inputs must be provided with a larger gap between them. Also as different parameter setups of the system will take different lengths of time, it is better to have an overestimate.

The delay chosen to the last layer is suitable for the experiments carried out in this thesis. It is possible that the value used here will be unsuitable for Neural Pipelines with more layers, because the delay is multiplied by the number of layers so with larger numbers the potential for error is higher. This is an important aspect to test before increasing the size of the architecture. It is discussed in the future work section 4.9.1.

The weight on this inhibitory connection is set to be 10 times larger than the inhibitory weight between the other layers. This is because the number of spikes in the input is much lower than the number of spikes in the other layers. As the input size and the activity in the layers is different in different simulations it is not possible to find a precise value. 10 spikes on input is typical this tends to produce upwards of 100 spikes in the layers. This is the reason that 10 was chosen for the multiplier. If the inhibition was taken directly from one of the layers then the multiplier would not be needed.

The inhibitory connections to the final layer are fully connected from the spike generator that provides the system input. Full connectivity was considered a suitable choice, because the layer must be completely quiet before the next input. It is the same as for the other inhibitory connections.

It is important to note that no attuning has taken place with these parameters. These values were chosen for the reasons outlined above and they work correctly for the architecture as it has been tested. Once determined the chosen values were not altered for any of the experiments, which shows robustness. The values do not need to be precise, as with the other parameters they must be balanced. For example if the connectivity was decreased then the weight value should be increased to maintain the same overall level of inhibition.

4.6 Internal Layer Parameters

The connections within each of the layers are known as the internal connections. The internal connections have the same choices for parameters as the external connections: level of connectivity, weight and delay. In each layer there are both positive and negative connections and there are two different configurations of these. Each experiment uses either one of the configurations. The first configuration is to have every neuron with a mixture of excitatory and inhibitory connections from it to other neurons. The second follows Dale's principle, so each neuron only has connections that are either excitatory or inhibitory going from it to the other neurons. This is explained in more detail in section 2.2.3.

The following sections describe how the weight, delay and connectivity values were chosen for the Neural Pipeline architecture. They also illustrate how these parameters can be changed to influence the system behaviour.

4.6.1 Weights

The Neural Pipeline architecture has both excitatory and inhibitory weights within the layers. Early versions of the architecture had only excitatory weights, but inhibitory weights were introduced to reduce the level of spiking within the layers. As seen earlier in section 4.5.1 high levels of spiking are a trait of under inhibited behaviour, so are undesirable. The distribution of the weights on the connections within the layers determine whether the network follows Dale's Principle. When they are distributed so any given neuron has only positive or negative weights on its output connections, then the network follows Dale's Principle, otherwise they are mixed. In all tests where mixed neurons are used each neuron has an equal number of positive and negative output connections. All of the tests using Dale's principle have used half excitatory and half inhibitory neurons. Half of each type of neuron was chosen because it allows a balance of excitation and inhibition. It was determined that this parameter would not be investigated, because the weights on the excitatory and inhibitory connections could be altered instead of the number of each type of neuron. Varying the number of each type is a possibility discussed in the further work section 4.9.1.

Some tests have used randomly chosen internal weights within a range of values. Others use fixed values for all connections of plus or minus the same value. As the randomly chosen values are uniformly chosen the average value of these two sets of values is the same. The weights were fixed (to plus or minus the weight value) for many of the experiments so that the impact of the other parameters could be explored without randomised weights.

The graphs shown in figure 4.10 illustrate how the behaviour changes with different internal weight values. The connectivity is also varied in these graphs to show how this relates to the weights. The other parameters are fixed throughout, with 50 neurons per layer and 5 layers.

It can be seen by comparing figure 4.10 (a) and (c) that **the advantage of having a low internal weight value is that it is possible to have a wider**

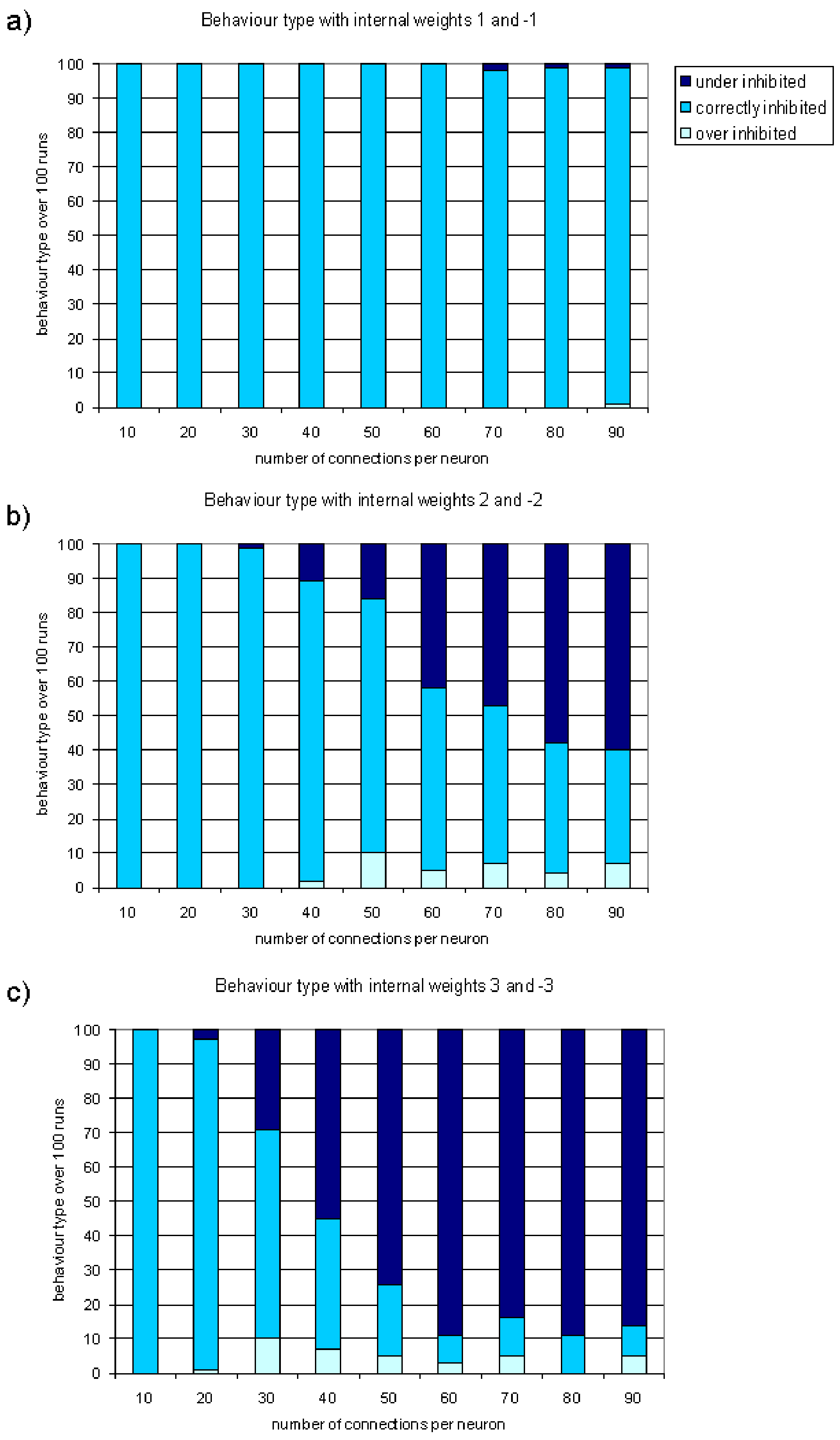


Figure 4.10: The type of behaviour found during 100 runs of each value of connectivity between 10 and 90, for layers with internal weight values of (a) 1 and -1, (b) 2 and -2 and (c) 3 and -3.

choice of number of connections per neuron with 100% correct behaviour. With larger weight values the overall number of correct runs is smaller, particularly at high values of connectivity. With a weight value of 3, only 10 connections per neuron has 100 correct runs, but with a weight value of 1, up to 60 connections per neuron still generated 100 correct runs. Therefore low weight values are more robust to variation in the number of connections. **This shows that even if the values chosen for the architecture are not optimal, choosing other values well can compensate. So here even if the weight value is high, having a lower value of connectivity can still allow the system to have a high probability of producing a correct run.** Conversely if the weight value is chosen wisely there is a large choice of connection values that will be likey to provide correct behaviour.

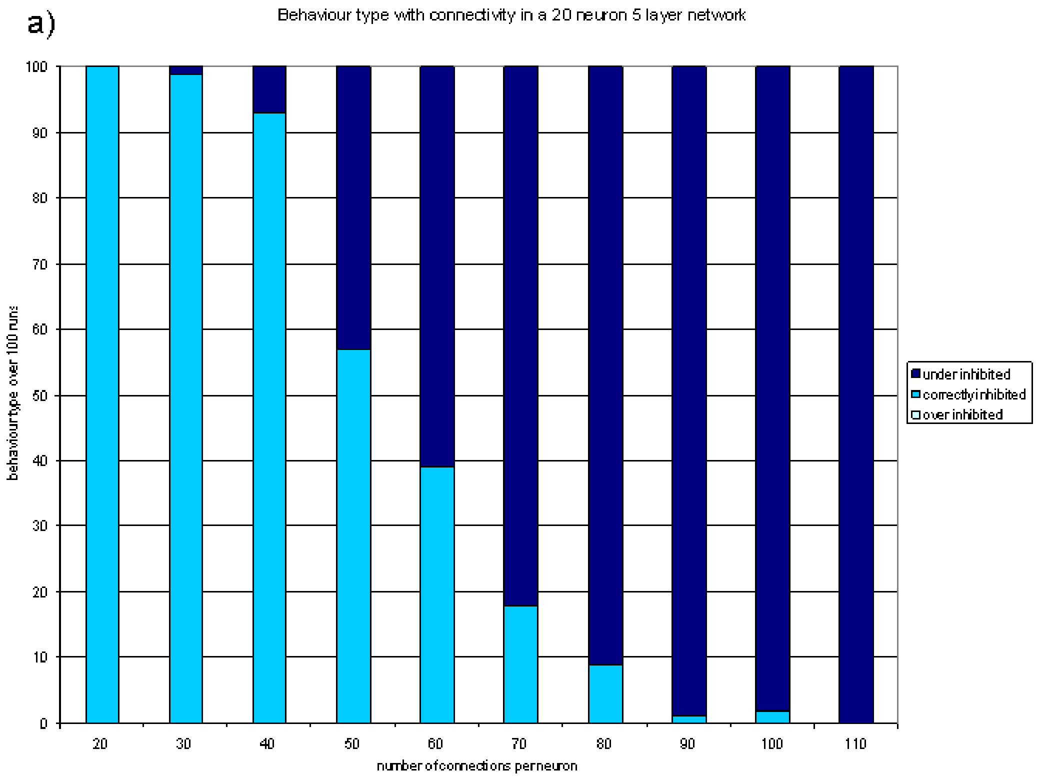
4.6.2 Delays

The delays on the internal connections have not been altered during the development of the Neural Pipeline architecture. This is because it is not necessary for these connections to have long delays, or different delays to one another in order for the architecture to perform correctly. 1ms was chosen as a fixed parameter for the delays, because it is a small delay and is the same value as on the external excitatory connections. If the application that the architecture is being used for requires delays within the layers, for example to improve the training then they can be introduced. For example the delays of a network are trained in [33]. This may require parameter changes elsewhere, but should not stop the architecture operating correctly. This is discussed in the future work section 4.9.1.

4.6.3 Connectivity

The connectivity within the layers of the Neural Pipeline architecture has an impact on the type of behaviour that the system exhibits. Preliminary tests here show how this changes with the size of the layer.

Figure 4.11 shows the results of 100 different simulation runs of each connectivity value for layers of (a) 20 and (b) 50 neurons. Figure 4.12 shows the same experiment with layer size (a) 100 and (b) 250. The graphs are plotted every 10 connections until there are no correct runs. **These graphs show that it is possible to achieve all 100 runs with correct behaviour for each of these layer sizes.** They also indicate that there appears to be an upper limit for connectivity that gives 100% correct behaviour. This appears to be at 40 connections per neuron. The transition was examined more closely for 100 and for 50 neurons, and the largest number of connections with 100 correct runs was found in both cases to be 42. The way that the other types of behaviour start to take over gradually, suggests that lower values of connectivity have a higher probablity of achieving correct behaviour, and is the reason for choosing a low connectivity value in the final architecture (chapter 3) and for the learning experiments presented in chapter 6. This connectivity constraint is not a biological one, it depends on the other network parameters used. Here the network is small, measured in hundreds of neurons compared to the billions of neurons found in the brain. For this reason the threshold of each neuron is set to be



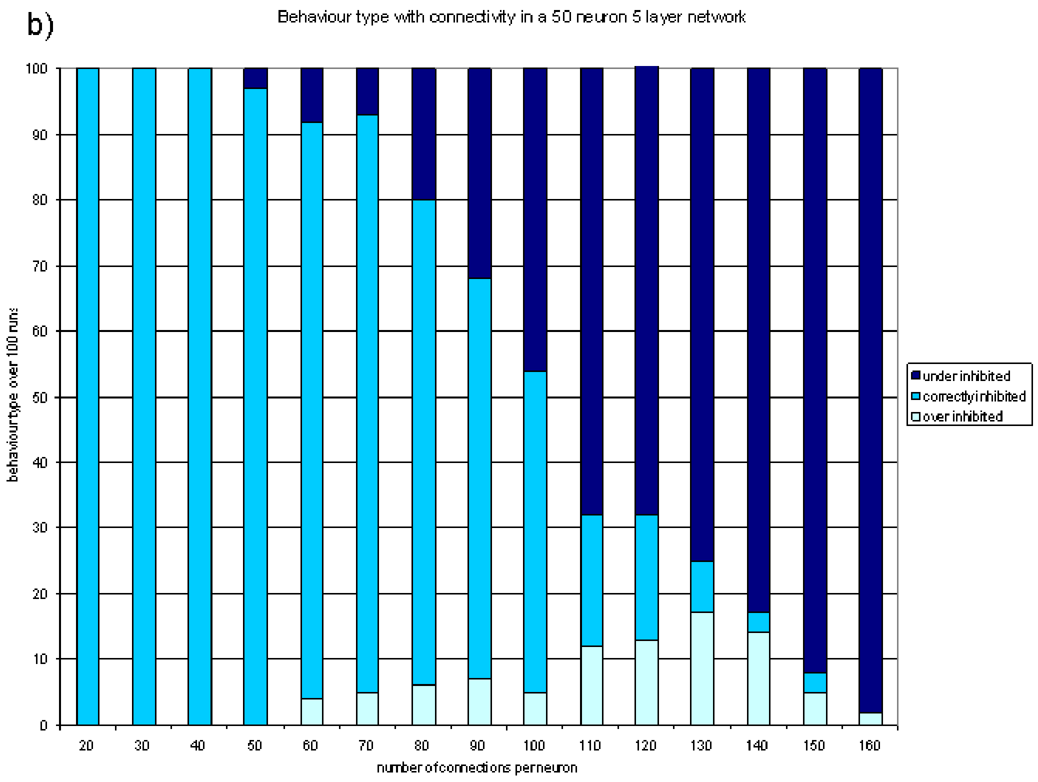


Figure 4.11: The type of behaviour over 100 runs of different values of connectivity with layer sizes of (a) 20 and (b) 50 neurons.

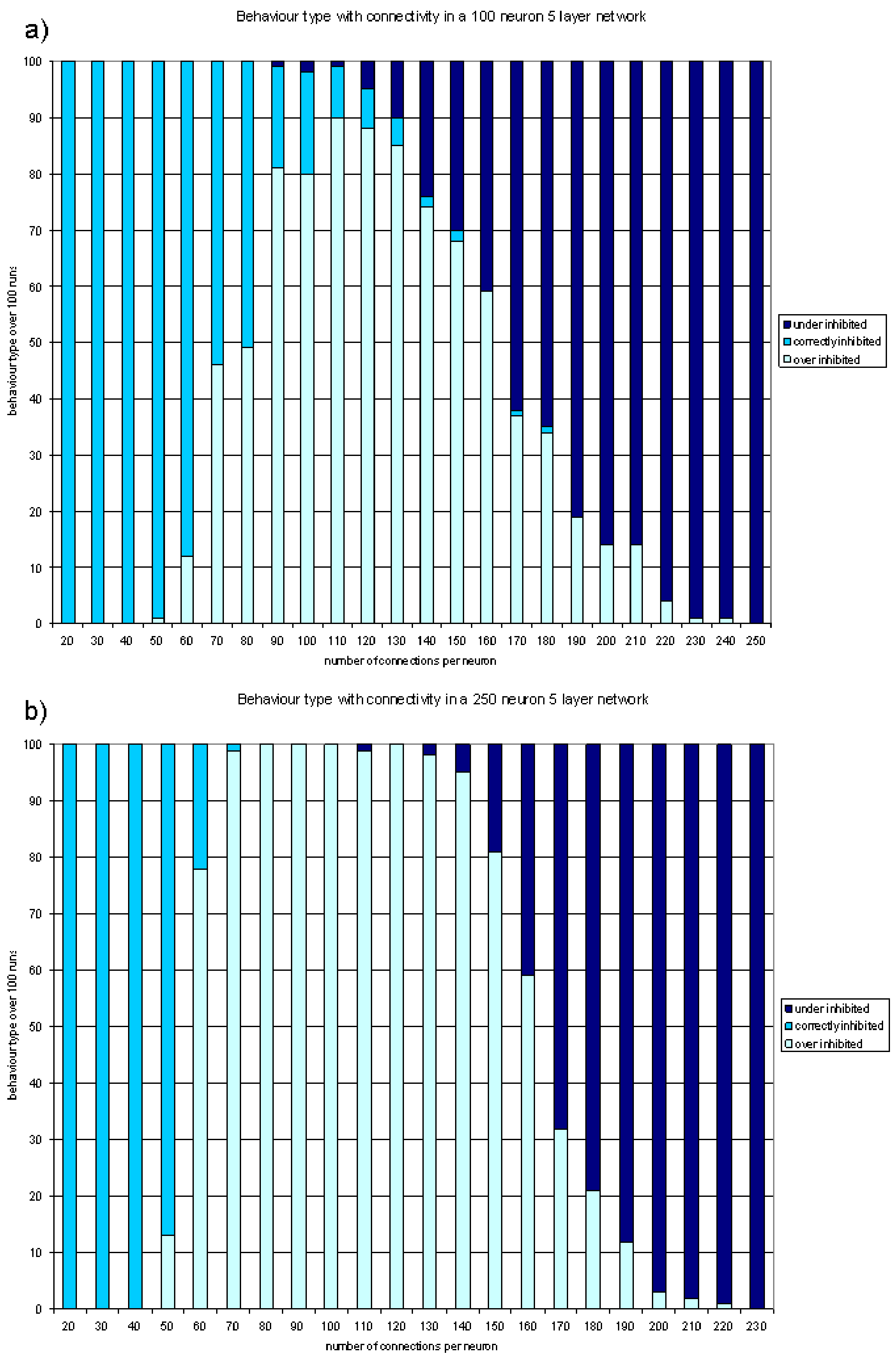


Figure 4.12: The type of behaviour over 100 runs of different values of connectivity with layer sizes of (a) 100 and (b) 250 neurons.

low so that a few input spikes are sufficent to produce an output spike. This means that the limit on connections for the architecture is low when compared to biology.

The smaller layer sizes have correct behaviour with a higher proportion of connections compared to their size. In the 20 neuron case with 20 connections per neuron (enough to be fully connected) all 100 runs show correct behaviour. With 50 neurons per layer and 50 connections 97 of them are correct. However in the larger sizes of network there are many fewer correct runs when the connectivity is the same size as the layer. Size 100 has only 18, and size 250 already has 0 correct runs by a connectivity of 230.

The region in which over inhibited behaviour is more frequent is much larger for the higher layer sizes. In figure 4.11 (a) there are no instances of over inhibited behaviour, as the connectivity is increased the system tends towards under inhibited behaviour. In the largest tested layer size of 250 neurons (figure 4.12 (b)) there is a region of 100% over inhibited behaviour between connectivity 80 and 100. This over inhibited region provides an area of compromise for these layer sizes. If for some reason the connectivity of the network had to be higher than 40 (where 100% correct behaviour stops) then while in an over inhibited region (between 50 and 100) the time between inputs could be increased to change the behaviour to correct (see definitions of behaviour for why this occurs 3.4).

Depending on the application, it may be acceptable to have an expectation of correct runs less than 100% of the time. If other factors dictate the layer size and connectivity then the simulation could be run multiple times and the output could be taken as the most frequently produced response. The system can be tailored to whether a correct response is most important, or the initial setup of the architecture.

The regions of behaviour are an interesting result, although the main result to be taken from these graphs is the usable region of correct behaviour. There are likely reasons for the other regions of behaviour, which are discussed here. These reasons are speculative, and have not been investigated further because only correct behaviour is desirable and these graphs were used to select a connectivity that provides a high level of correct runs. Suggested reasons are given here and ways to investigate whether they are true are suggested as further work (section 4.9.1).

The under inhibited region starts to appear as the connectivity is increased. With high levels of connectivity every neuron is connected to more of the other neurons. This means that every spike generated can produce more spikes, meaning that the activity level is higher than with lower connectivity. It is thought that the inhibition between the layers is unable to completely suppress this increased activity, so the behaviour becomes under inhibited.

There are two aspects that may suggest the switch to completely under inhibited behaviour is unexpected. One is that there are 50% inhibitory connections in each of the layers, so these neurons should prevent the activity getting too high. It is believed that the inhibition within the layers will limit how high the activity is able to become, but this limit is higher than the level of activity required to cause under inhibited behaviour. The random allocation of the connections may also have an impact on this, as although the average connectivity is the same, individual neurons may be very active and therefore difficult to inhibit. The second aspect is that the

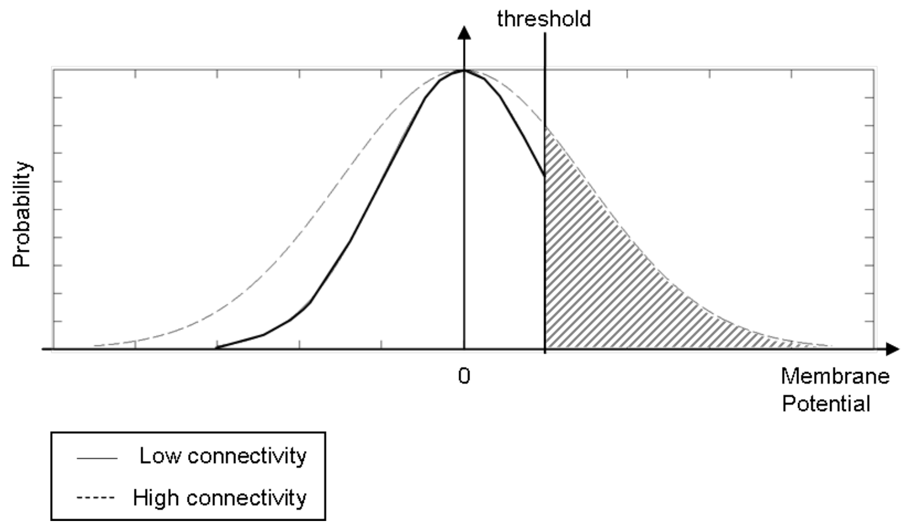


Figure 4.13: A graph illustrating the probability of a neuron having a particular level of membrane potential. A neuron is able to have a more negative value of membrane potential than positive. This is because the threshold voltage setting means that it is not possible for a neuron to be in the shaded region of the graph.

amount of external inhibition should increase too, as the activity in all layers should increase with the connectivity. It is thought that the delay on the inhibition means that the first layer can start spiking more heavily, so it is harder for the second layer to stop it.

The over inhibited behaviour is not seen at all for low numbers of neurons and it increases as the number of neurons is increased (see figures 4.11 and 4.12). With 250 neurons per layer (figure 4.12 b) there is a clearly defined region of over inhibition. This region appears at a higher connectivity than the correct region, but lower than the under inhibited region.

Two different hypotheses are proposed to explain the behaviour. They are suggestions to be tested in future work (section 4.9.1). The first relates to the over inhibited behaviour, so explains the transition for the 20 neuron case. The second may explain the two transitions with larger numbers of neurons.

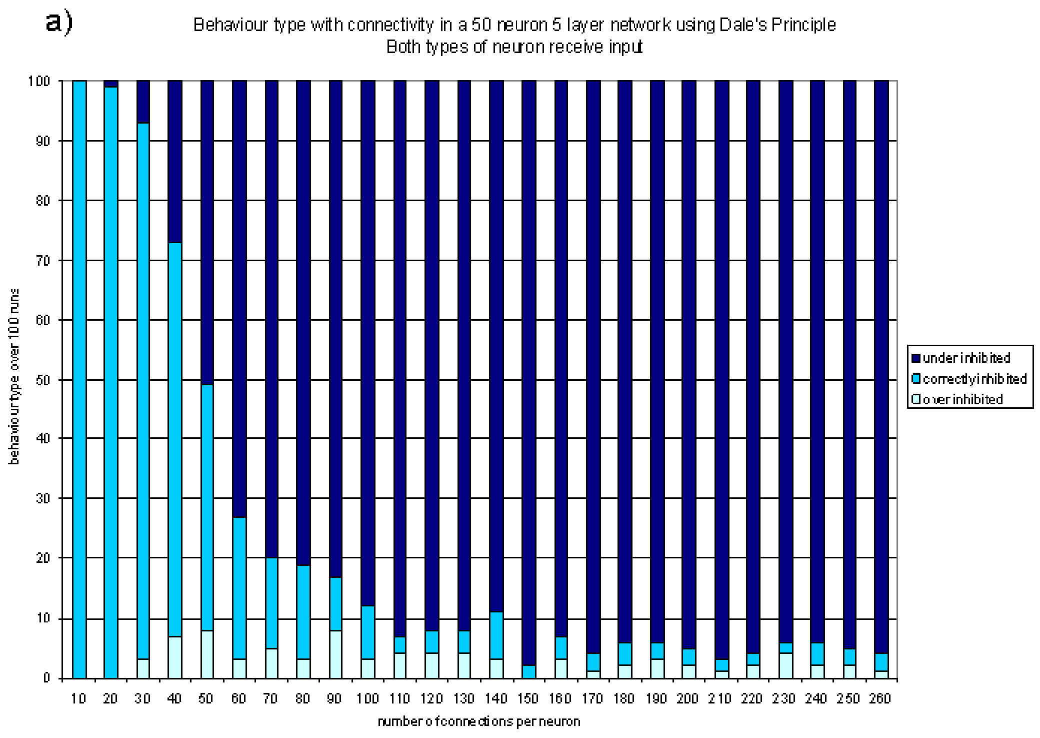
Hypothesis 1 is that as the amount of activity in each of the layers increases, the balance of the layer will become increasingly inhibitory. This is because the neurons are able to store a large amount of inhibition. Storage in this case relates to the value of the membrane potential, and how long it remains at the value. Figure 4.13 illustrates how the neurons are able to store inhibition but not excitation. Every inhibitory spike received as input will decrease the membrane potential of the neuron. This is different to the excitatory case, because as soon as the membrane potential reaches the threshold it is reset. This means that a layer can store inhibition over longer periods than excitation. As the connectivity of the layer is increased, more activity will be present, and the layer will have more neurons with a higher level of inhibition. This will make over inhibited behaviour more likely.

The second hypothesis relates to the multiplication provided by the weights on the external excitatory connections. This means that each layer has more spikes than the previous layer. It is suggested that when the system is operating in the correct behaviour region the balance of spikes in layers 1 and 2 is correct, so that 2 shuts off 1 but not so strongly that it cannot respond to the next input. In the over inhibited region it is proposed that the activity in layer 2 is 'too high' when compared to layer 1 so it stops the next input. Importantly in the over inhibited region the activity in layer 3 is not 'too high' when compared to layer 2, or it would stop this occurring. In the under inhibited region layer 3 is now 'too high' compared to layer 2. This allows the second input to pass through layer 1 but no further, because layer 2 is being strongly inhibited. This means that layer 1 can continue spiking, uninhibited, and cause under inhibited behaviour.

4.6.4 Dale's Principle

Dale's Principle [20] (described further in section 2.2.3) states that almost all neurons are either excitatory or inhibitory. The neurons in the earlier tests of the Neural Pipeline architecture described above (figures 4.11 and 4.12) use neurons that have both excitatory and inhibitory connections. To make the architecture more similar to a biological neural network, Dale's Principle was introduced. The impact of this upon the system behaviour can be seen in figure 4.14 (a) when it is compared with figure 4.11 (b). It is possible to see that there are several differences between the two graphs. Firstly **the region with 100% correct behaviour is shorter in the system that uses Dale's Principle.** Only connectivity 10 has 100 runs with correct behaviour, as opposed to up to connectivity 40 without. This alone would make it seem that the system using separate excitatory and inhibitory neurons is less desirable than the mixed neuron version. However when the rest of the graph is considered, **when Dale's Principle is used there are some (although a low number of) correct runs for all of the connectivity values tested.** When mixed neurons are used, the correct runs stop at 150 connections. This observation is consistent with the earlier comparison of mixed and Dale's neurons for different external inhibitory weights in figure 4.7. In that case too Dale's principle neurons can produce correct runs at more extreme values than mixed neurons. The system using Dale's Principle has fewer over inhibited runs per column, but these too continue for all of the values of connectivity after the correctly inhibited region.

It is likely that the continuation of correct runs at large connectivity values is due to the more extreme possibilities of connection when using Dale's Principle. By this it is meant that when mixed neurons are used the extreme values tend to be averaged out, because each of the neurons has equal numbers of excitatory and inhibitory connection. **With Dale's Principle it is more likely that a neuron can extremely excite or inhibit other neurons. This means that for some of the runs the activity is kept at the right level for correct and over inhibited runs.** The reason that 100% correct behaviour finishes at a lower connectivity value is probably similar. The more extreme values make it likely that some of the runs with low connectivity will be incorrect. As with the results for the system without Dale's Principle, this reasoning is speculative and has not been tested here, because



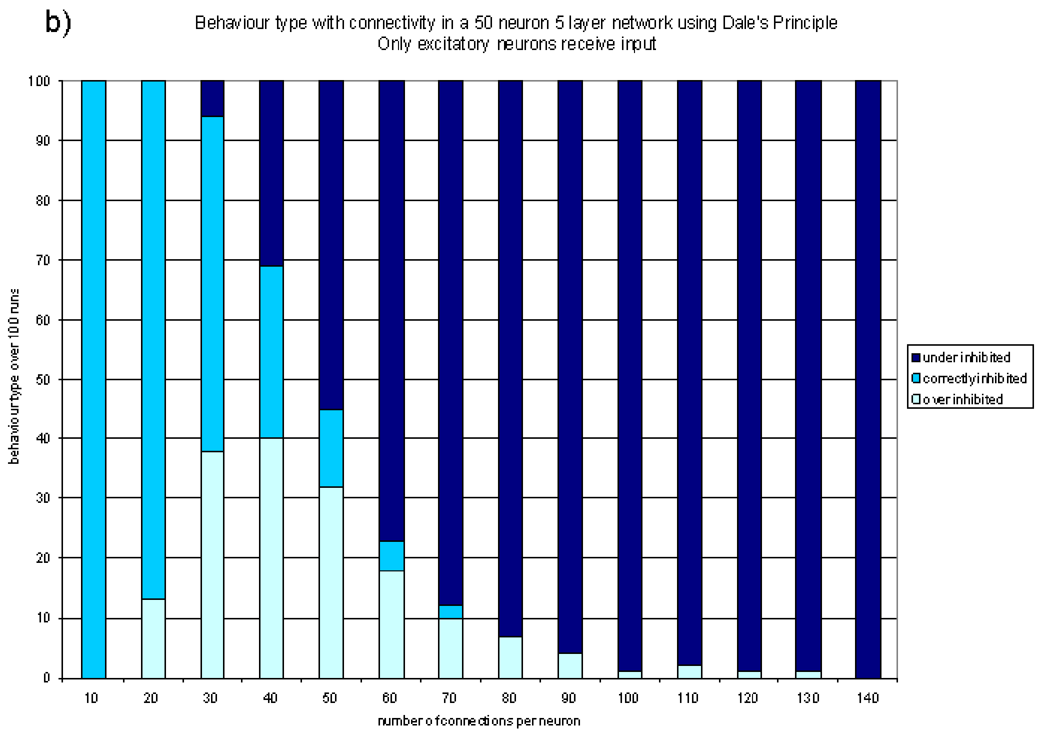


Figure 4.14: The behaviour over 100 runs for different values of connectivity when neurons that follow Dale's Principle are used. Graph (a) shows the situation when the input is presented to both inhibitory and excitatory neurons and graph (b) shows the case when the input is only presented to the excitatory neurons.

the correct region is the area that has been concentated on. Instead it represents another possibility for future work (section 4.9.1).

Neurons that follow Dale's Principle were chosen for the Neural Pipeline architecture, because they are more biologically realistic and because it had been shown here that with low connectivity values it is possible to achieve 100 correct runs. The ability to find a number of correct runs even with very high values of connectivity is also considered to be an advantage, although there is a trade off with the size of the region with all correct runs.

The result when Dale's Principle is used, but the stimulus is provided only to excitatory connections is shown in figure 4.14 (b). There are clear differences between this and the case when the input is presented to both excitatory and inhibitory neurons (figure 4.14 (a)). The correct behaviour is all in the region of 70 connections per neuron or fewer, there are no correct runs at higher levels of connectivity. There are more over inhibited runs in the over inhibited region in graph (b) than graph (a). The drop off in correct behaviour is worse when only excitatory inputs receive a stimulus.

It is thought that when the stimulus is presented to just the excitatory neurons there is much more stimulation initially. This will mean there is a much lower chance of correct runs occurring, because the inhibitory neurons do not get the chance to keep the activity within the layer low enough for correct behaviour. It is for this reason that the input stimulus in the Neural Pipeline architecture is presented to both excitatory and inhibitory neurons.

The connectivity is varied at the same time as the external inhibition to identify how the parameters interact. The result of this is given in figure 4.15. The graph shows that **the connectivity has much more of an impact than the external inhibition for the values tested.** With a high level of inhibition there are more correct runs, but the variation between the highest and lowest values is small. The low connectivity levels perform consistently well independently of the inhibition. With very high levels of connectivity a slight improvement is found by using larger levels of inhibition. This suggests it is more important to choose a good value of connectivity.

4.6.5 Number of neurons in a layer

The number of neurons needed within each of the layers is influenced by the size of the input required. This is particularly important if the each input is presented to a different neuron to avoid compression, as is the case in the experiments presented here. This means that the size of the input is the minimum layer size. If the input can be compressed then this restriction does not apply.

Larger layer sizes allow larger inputs, however there is a limit to this depending on the settings of other parameters, particularly the external inhibition. It shows the importance of using reasonable values for each of the parameters, rather than a 'perfect' value for just one of them.

When considering the impact of layer size on the behaviour of the system, 100 runs of layer sizes from 10 to 100 were run. This used an early version of the

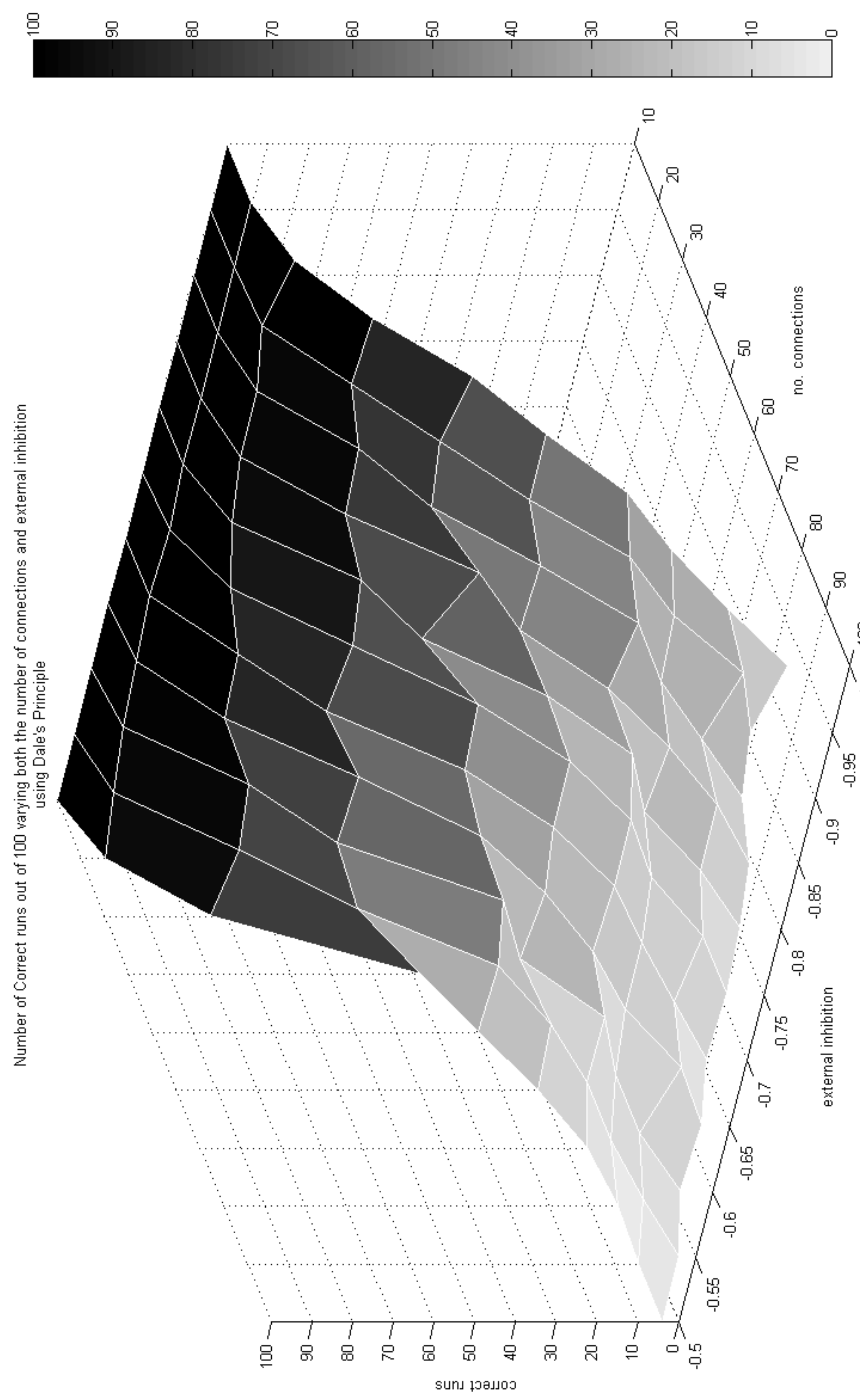


Figure 4.15: The number of correct runs out of 100 for each of the different parameter settings, when varying both the external inhibition and the internal connectivity. The system uses Dale's Principle neurons.

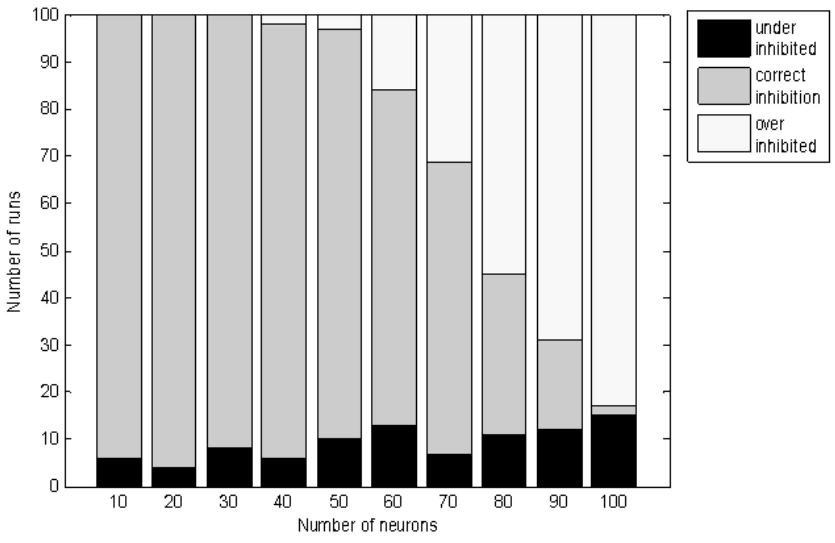


Figure 4.16: The type of behaviour exhibited by the system, when using different numbers of neurons in the layers. All layers have the same number of neurons and 100 runs of each layer size are run.

architecture with connectivity of the same size as the layer. The results of this test are shown in figure 4.16. The number of correct runs drops steeply as the number of neurons is increased above 60. This result, however, is actually thought to be because of the connectivity of the layers. The results presented in the previous section (4.6.3) show why this is the case, as it is shown that networks with layers of 250 neurons can exhibit correct behaviour.

The input for the system was the same throughout the experiment, with 2 neurons receiving the input. This meant that the proportion of the neurons in the layer that received the input varied depending on the size. To make sure this did not influence the results a scaled input test was carried out, with 10% of the neurons receiving the input. The results of this can be seen in figure 4.17. This graph shows the same **pattern of reduced correct runs with increasing numbers of neurons** as figure 4.16 shows with the unscaled input. Therefore the argument that **the connectivity is the cause of this behaviour** (section 4.6.3) is not disproved by this.

There are some differences in the two graphs, firstly there is no under inhibited behaviour when the input is scaled, and secondly the behaviour stops being correct at lower numbers of neurons with a scaled input. This is suggestive that there is a limit for input size that will provide correct runs.

The number of neurons in a layer also has an impact on the transition between behaviour types with varying connectivity. This is shown in figures 4.10, 4.11 and 4.12 and is described in the previous section. With low numbers of neurons there is no over inhibited region, but as the number of neurons is increased the over inhibited region increases in size.

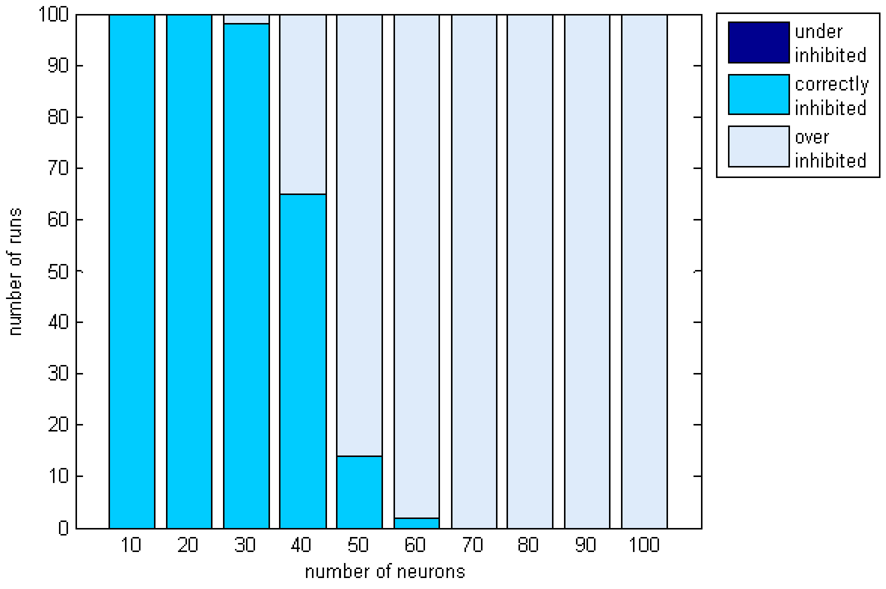


Figure 4.17: The type of behaviour exhibited over 100 runs for each layer size, when the stimulus size is scaled with the layer size.

The size of the layers is one of the most varied parameters in the experiments carried out. Layer sizes of up to 10,000 were tested to identify the largest input size for different sizes of layer with different internal weight values, with layers as small as 5. Other experiments in this chapter use different sized layers with 50 and 100 being the most frequently used.

Throughout the experiments presented here each of the individual Neural Pipeline layers is set to the same size as one another. This was a decision taken for simplicity, and because introducing different layer sizes in one Neural Pipeline introduces a combinatorial choice. As there are too many parameters to investigate fully during the scope of this work this is one option chosen not to be investigated. It does provide another possibility for future work as described in section 4.9.1.

4.7 Input

The input that the system is required to process also influences the parameter choices for the Neural Pipeline. The experiments presented here are concerned with the throughput of the system and with the input size. The throughput is considered by looking at how long it is necessary to wait between inputs. The input size is the number of input neurons that can be activated simultaneously. The parameters that must be changed to allow larger inputs are discussed.

For the experiments presented in this thesis the inputs are presented and then stopped after 10ms. With the exception of the experiments with a noisy input in section 6.5, the spikes appear at regular intervals of 1ms. This level of spiking was chosen because it provided sufficient activity in the first layer to see correct behaviour.

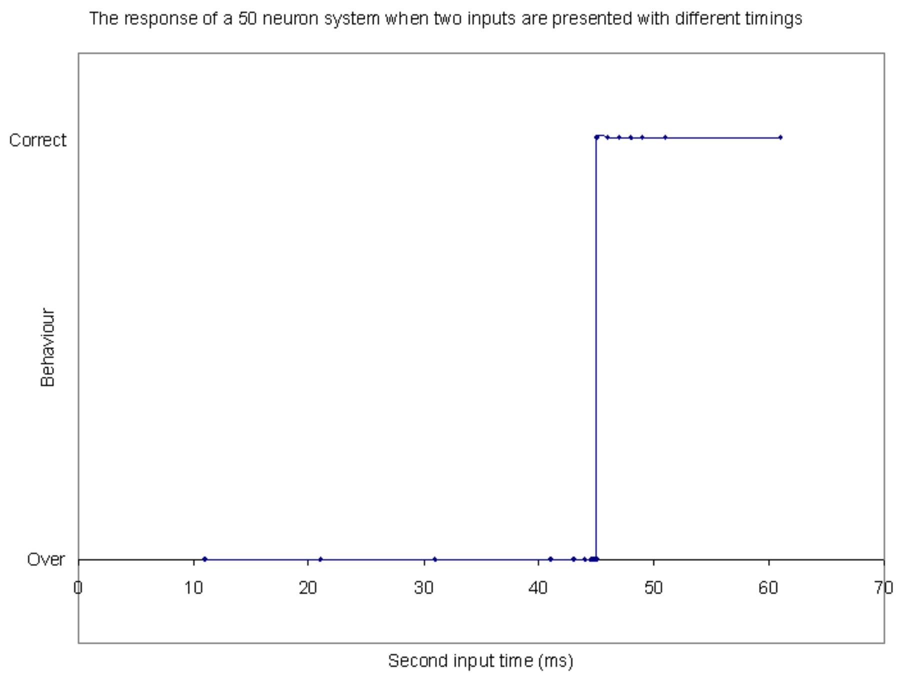


Figure 4.18: A graph to show the type of behaviour found with different lengths of time between two inputs.

An extension to this proposed for future work (see section 4.9.1) is to provide the system with a continuous input. The architecture should then break up the input from a continuous stream into blocks, using the external inhibition. This may be a useful extension, because splitting the continuous stream permits the output to be given at a particular time.

4.7.1 Time between inputs

The throughput of the system varies depending on the parameters that are used. The initial experiments presented here show that **the crossover between correct and incorrect behaviour has a sharp cut off**. With a layer size of 50 neurons, the switch occurs with a 45ms gap between inputs, lower than this the behaviour is over inhibited and higher than this the behaviour is correct. This is shown in figure 4.18. This shows a well defined limit for how long the gap between inputs must be.

4.7.2 Size of input

The input size, in these experiments, is taken to be the number of neurons in the first layer that receive an input. Larger inputs require a larger layer size, if each input is to be presented to a different neuron on the first layer. This is used for all of the experiments presented here, to avoid the input being compressed.

Larger inputs can also provide more excitation in the first layer, this means that some of the parameters can be adjusted to promote correct behaviour for different sizes of input. This is shown for the internal weight values in figure 4.19. There are 200 neurons per layer in this test and different input sizes are used with each internal weight value, (a) has the highest internal weight value of 1 and (d) the lowest at 0.1. With a weight of 1 the highest input size where all 100 runs are correct is 24, but once the weight is dropped to 0.25 the system can handle 45 inputs with all 100 runs behaving correctly. A weight of 0.5 results in 100 correct runs for 30 inputs, between these two values. It appears from these results that 45 may be a limit (or approaching a limit) for a network using these parameter values, even if the weight is reduced. This is backed up by the results shown in table 4.1, where even with layer sizes of up to 10000 neurons, with weights as low as 0.001, the highest number of inputs with 100 correct runs is 45.

To allow larger numbers of inputs to be presented to the system the external inhibition can be lowered. With a lower inhibition level the system does not become over inhibited as quickly with large numbers of inputs. With an inhibitory weight of -0.3 it is possible to have 81 inputs with a layer size of 81 neurons. This is the largest input size that has been tested, because it is the size of input required for the learning experiments in chapter 6. However, adjusting the parameters further should allow larger inputs still.

No. of Neurons	Weights	Highest 100% Correct Inputs
200	0.25	45
200	0.1	45
200	0.05	45
500	0.5	25
500	0.1	45
500	0.01	44
500	0.005	44
1000	0.005	44
10000	0.001	44

Table 4.1: The highest number of inputs with 100 correct runs for different layer sizes.

Another point to note from the graphs presented in figure 4.19 is that as the internal weights are decreased the switch in behaviour from correct to over inhibited is much more sudden. With weights set to 1 (graph a) there are several sizes of input with some correct runs and some over inhibited runs, but when a weight of 0.25 (graph b) is used, the transition is from 100 runs correct to 100 runs over inhibited. This suggests that changing some parameters can make the other parameter choices more sensitive.

For other parameters, the transition between behaviour types is less clear, for example when internal weights of 0.5 are used. In this case the number of correct runs starts to decrease, then increases sharply before decreasing again. The graph is shown in figure A.8 (b) in the appendix. This is not considered to be a problem, because the parameters are chosen to be within the correct region, not near the boundary. Also this particular example may be an artefact of the simulation environment being

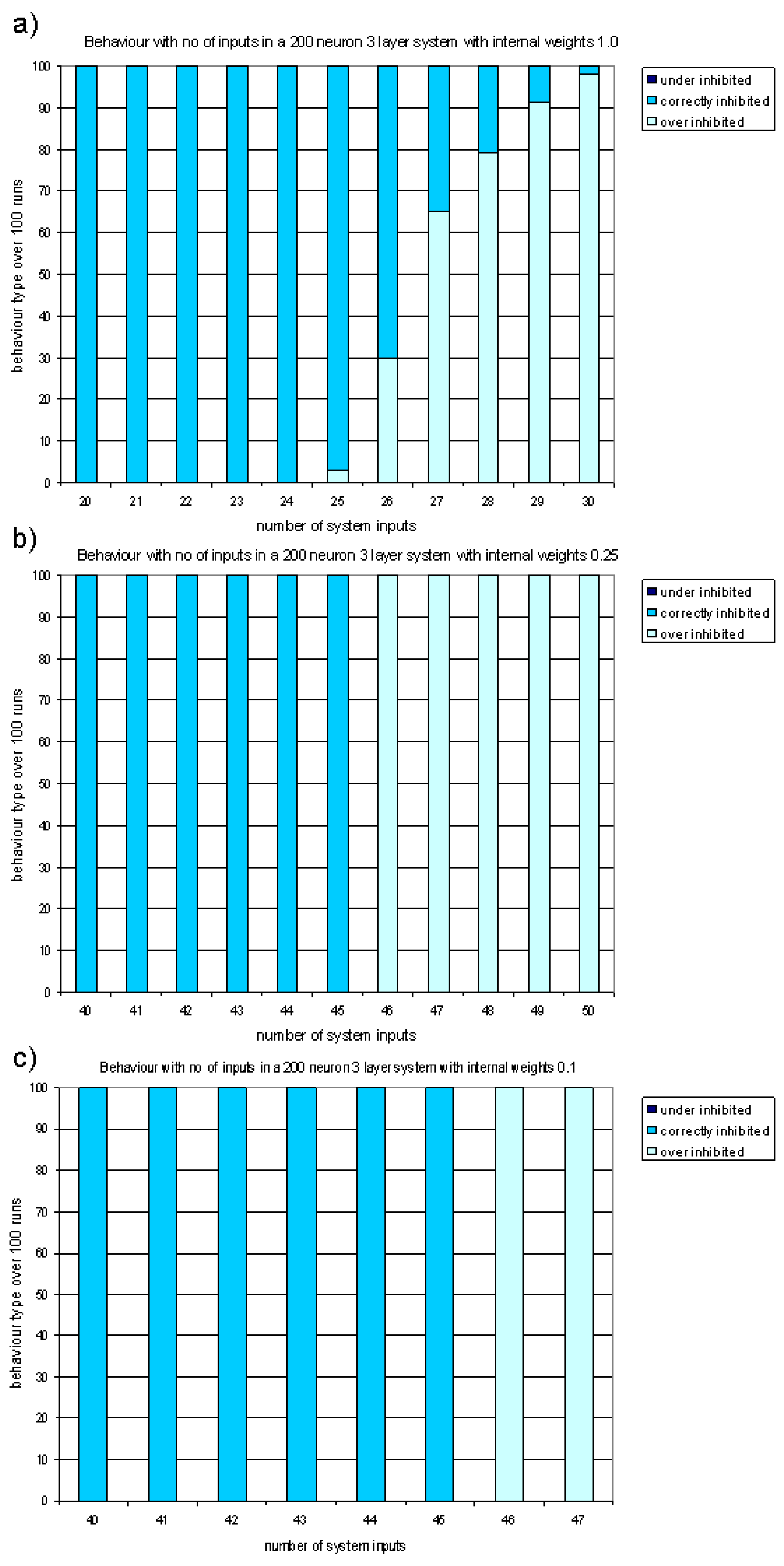


Figure 4.19: The behaviour type for 100 runs of each number of active inputs, with different internal weight values. (a) has a internals weight of 1, (b) 0.25 and (c) 0.1.

used, because internal weight values either side of this (0.49 and 0.51) show a smooth transition. These graphs are presented in figure A.8 in the appendix.

The size of input was also investigated with regard to the layer size in section 4.6.5. Figure 4.17 shows that there are no correct runs with over 70 neurons with 7 inputs, whereas in figure 4.16 over 50 of the 100 runs are correct with 70 neurons (and only 2 inputs). This suggests the same response as the results presented in table 4.1, that there is a limit for the size of input, even as the layer size is increased. The limit in the case of figure 4.17 is lower than in the table, but this is because the architecture was not as developed in 4.17 and used different parameter settings. It still provides additional evidence for the limit of input size, with the weight values constant.

4.8 Pieron's Law

A biological trait that falls naturally out of the operation of the system is Pieron's Law. Pieron's Law describes the relationship between stimulus intensity and reaction time. This relationship is given in equation 4.1 where RT is the average reaction time, R_0 is the minimum possible reaction time, I is the stimulus intensity and k and β are constants. Figure 4.20 gives a generalised graph of Pieron's law.

$$RT = R_0 + kI^{-\beta} \tag{4.1}$$

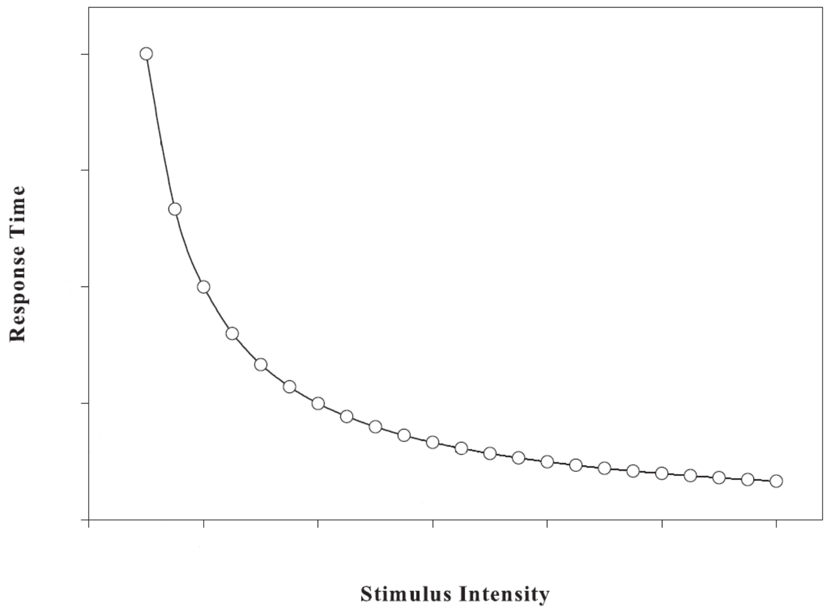
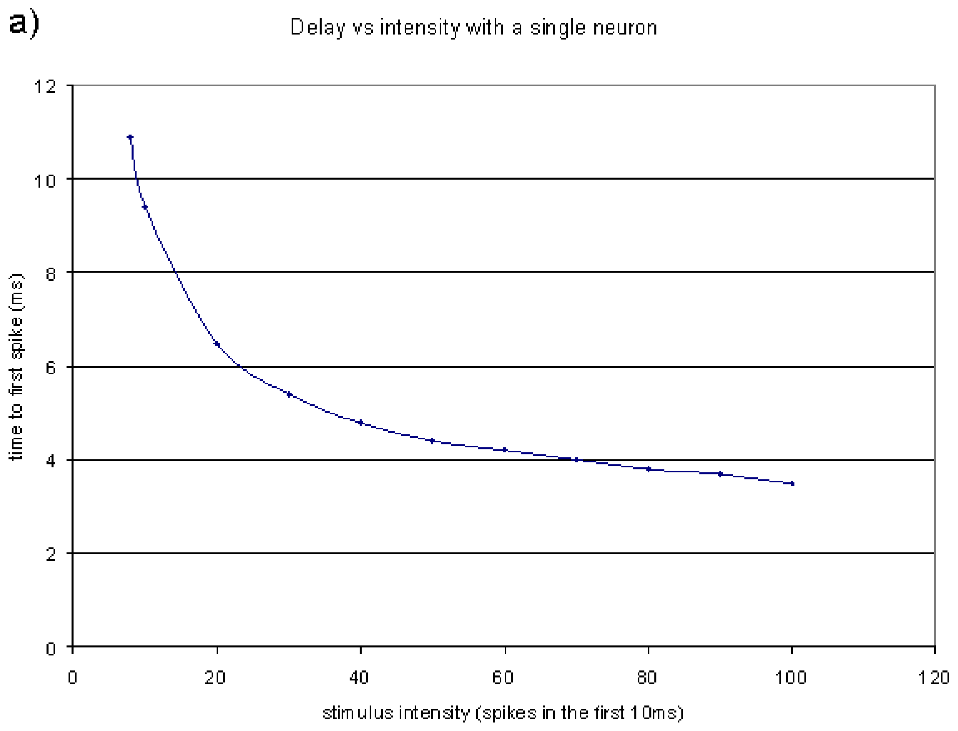


Figure 4.20: A graph of Pieron's Law adapted from a more specific graph in [62].

It is perhaps unsurprising that an integrate and fire neuron follows Pieron's Law, because the larger the number of inputs (or the larger the input weight) the sooner



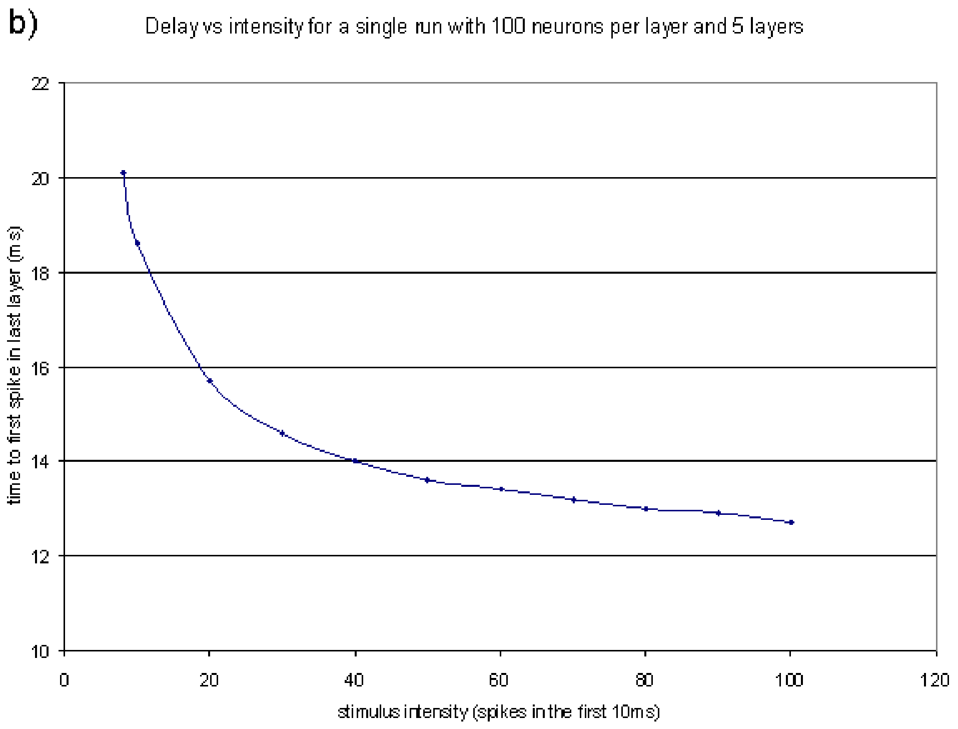


Figure 4.21: Graph (a) shows the amount of time a single neuron takes to fire when the stimulus intensity is varied. Graph (b) shows the time that it takes for layer 5 to produce a spike with different stimulus intensities in a Neural Pipeline with 5 layers.

the membrane potential reaches its threshold. There will be a limit to the speed of the response, based on the smallest possible rise time to the threshold. This was tested using a single LIF neuron in NEST and the result is shown in figure 4.21 (a). When this response is compared with 4.20, the graph shape is the same in both. This suggests that Pieron's Law holds for LIF neurons. It does not necessarily follow that Pieron's Law is applicable in a Neural Pipeline. To test this for a five layer Neural Pipeline the frequency of the input was varied and the first spike response from the last layer was measured.

The result of a single test is shown in figure 4.21 (b). A comparison of figures 4.21 (b) and 4.20 shows that the pipeline is consistent with Pieron's law for this example. Taking the average over five input stimuli and five different setups produces the same graph shape. Varying the number of layers does also. All of these graphs can be found in the appendix in figures A.9 and A.10.

4.9 Discussion

The experiments presented in this chapter demonstrate how the behaviour of the Neural Pipeline architecture is influenced by the different parameters. The various tests have also justified the design decisions that were taken during the development of the Neural Pipeline architecture.

The parameters can be broken down into: external inhibitory connections, external excitatory connections, internal parameters and the input. Both sets of external connections have weight, delay and connectivity. The internal connections also have these parameters with the additional parameter of 'number of neurons'. The input can be varied by altering how many neurons receive it, and by altering the time delay between inputs.

Each parameter influences the behaviour to a different extent and it is the balance between these parameters that produce correct behaviour. The key parameters that have been found to alter the system behaviour are the external inhibitory weight, the internal connectivity and the size of input. These parameters control the activity seen within the layers and the inhibition used to stop this activity. The results show how the parameters interact, so that if a particular value is required for one of them then the other parameters may be altered to produce correct behaviour.

To increase the chance of correct behaviour a low value of internal connectivity should be used. A value of 20 connections per neuron gives 100 correct runs out of 100 for sizes of layer between 20 and 250. As the size of the layer is increased it is possible to increase the connectivity and still achieve 100% correct runs. When Dale's principle is introduced, a connectivity value of 10 is required for 100 correct runs. This is the connectivity value that has been used for the Neural Pipeline architecture. The low connectivity values also perform consistently well with different external inhibitory weight values.

When the external inhibition is investigated, high values of inhibition result in over inhibited behaviour and low values in under inhibited behaviour. This is expected from the definitions of over and under inhibited behaviour. In the experiment which investigates external inhibition only a small proportion of the results are correct. This is a limitation of the other settings for this experiment and not a limitation of the system, because it is demonstrated in other tests that these values of inhibition can produce 100 runs of correct behaviour out of 100.

It is found that the size of input can be increased by increasing the number of neurons within a layer. There is a limit to this while leaving the other parameters fixed. With external inhibition set to -0.75 (as identified in section 4.5.1) 45 is the maximum number of inputs to produce 100 correct runs. However, setting the external inhibition to -0.3 allows a larger input size to be used (here 81).

The balance required is between the amount of activation in the layer and the amount of external inhibition. Many of the parameters, including the size of the input and connectivity, contribute to the activity within a layer. When there is more activity within a layer then there needs to be correspondingly more inhibition to switch it off. It is this activity level that determines whether behaviour is correct, with too much activity the incorrect behaviour types become more likely. This is seen by increasing the size of the input or the connectivity.

The weight values and threshold values of the neurons are likely to have the same type of impact as the size of input and connectivity, because effectively they are all controlling the level of activity within the layer. These parameters were not chosen to be tested, so are addressed in the following section.

All of the tests presented here use 100 runs for each parameter setting, so where there is 100% correct behaviour it means that 100 runs were correct. It is possible that if more runs are examined there will be some incorrect responses. The results reflect the probability of getting a correct run rather than a certainty. The trends they show increase the likelihood of choosing values that will provide correct runs. For example, low values of connectivity tend to produce more correct runs.

The results presented in this chapter illustrate that there is a wide range of parameter values in which the architecture produces a correct response. If one parameter must be set to a particularly high or suboptimal value, then the other parameters can be adjusted so that the architecture performs correctly.

4.9.1 Further work

The tests presented in this chapter have cover the parameters that were considered to have the largest impact on behaviour. Due to time constraints other parameters have not been investigated (or not investigated extensively) in the development of the architecture. These include the threshold of the neurons, the external weights, the internal delays and the internal weights. The influence of changing these parameters should be examined.

Rather than vary each of the parameters individually and compare their effects, an estimate of their influence on activity within a layer could be produced using a model that approximates the system. The parameters could be balanced using this model. It would not be possible to find the exact level of activity in the layer because of the random connectivity. This is not necessary, however, to balance the parameters.

The tests on the architecture consider layers processing time relative to one another to determine the behaviour type. The absolute time that it takes for the architecture to process is not examined. When the architecture is used for real world tasks the time taken is important, with a shorter time being advantageous. It is the summation of the delays on the external connections that dictates the time that the architecture takes to perform a task. The shortest time should be identified by reducing the delays until the behaviour is no longer correct. The delay could appear on the external excitatory connections, the inhibitory ones, or both. Different choices of delay set up may be preferable for different tasks and these should be established.

In the current architecture the delay to the last layer is set so that an input will provide a signal that will inhibit its resulting activity. This is because the delay is set to be longer than the time it takes for the activity to reach the last layer. It is possible to allow the last layer to continue to remain active until the next input is presented to the system by reducing the delay. This may be advantageous if the output from the system is still required, but the information content degrades over time. The extent of this degradation should be investigated to identify the limitations of this shorter delay length.

The delay to the last layer is multiplied by the number of layers so that it scales with the system size. The delay length has been suitable for this work using fairly small numbers of layers (three or five). It is important to test the suitability of this delay when carrying out tests with larger numbers of layers. In these cases the delay length is increased, so if there is a slight error with low numbers of layers then it will be made bigger.

Two hypotheses for the different regions of activity were presented in section 4.6.3 should be tested. If hypothesis 2 is correct then it suggests a way to improve the extent of the correct region by reducing the weight value on the external excitatory connections to reduce the level of multiplication in each layer. The reasoning suggested for the change in this behaviour when Dale's Principle is introduced (section 4.6.3) should also be investigated.

The influence of different numbers of neurons in different layers on the system behaviour should be investigated. It may be that by reducing the number of neurons in later layers, under inhibited behaviour could be converted to correct behaviour, by limiting the excitation within the layer. This might also be achievable by varying different parameters in the different layers, so that the activation level is controlled independently for each layer.

The input to the system is a factor that should be explored. In these experiments the input is presented for a burst and then stopped, but understanding the system's response to a continuous input will be important for applications. The layered structure of the architecture should split the input into bands of activity to be processed. This may mean that some portions of the input would be ignored, which may reduce the system's ability to respond correctly to inputs. An alternative form of the architecture which may be more appropriate for continuous input is suggested in section A.2.

4.9.2 Summary

The key to increasing the probability of correct behaviour is to balance the internal layer activity with the external inhibition. The activity is controlled by a number of parameters including the internal weights, connectivity and input size. They can be adjusted to correctly balance with the external inhibition. Typically, low values of connectivity are preferred, to keep the activity low. The number of neurons in the layer is also important, for larger inputs the number of neurons needs to be increased. This may need coupling with a reduction in external inhibition, depending on the input size.

This chapter has explored the influence of the different parameters on the system behaviour, but there is still variance between simulations with the same parameter settings. The following chapter analyses the difference in behaviour between these simulations. The use of parameters to controlling the system behaviour allows suitable choices to be made for the architecture for the application of learning. This is described in chapter 6.

Chapter 5

Analysis of Behaviour

5.1 Introduction

In this chapter the Neural Pipeline architecture is analysed in two ways. The first relates to the system behaviour; the architecture is able to demonstrate three fundamental types of behaviour. The analysis investigates the system properties that cause different simulation runs to exhibit different types of behaviour. The second is an investigation into how different inputs can be identified by the system, and how the layers influence this identification.

The behaviour of the system can be altered by changing the system parameters as identified in chapter 4, but it is also shown that the behaviour changes between different simulation runs using the same parameter values. The only changes between simulation runs are the randomly chosen internal layer connections. Different properties of these connections are considered to identify their influence on the behaviour. These properties are the number of self connections, the value of the weights on each neuron's inputs and the number of input connections that each neuron has. The difference in each property is examined for examples of the three different types of behaviour. From this it is suggested that extreme weight values on individual neurons contribute to incorrect behaviour.

In order for a layer to be able to identify inputs, when different inputs are presented to the system the internal response should also be different. Each input should cause different neurons in a layer to fire at different times. Analysis of the distinguishability of two different input signals is performed using a metric which compares the number of times each neuron fires for different inputs. Using this metric it is possible to identify how identifiable the signals remain over time and through the different layers of the architecture. This provides an indication of how easy it is for the system to be trained on sets of inputs. Inputs with more distinct responses within the layers are easier to train the system on.

5.2 Behaviour Analysis

As correct behaviour is the desirable behaviour type it is beneficial to identify a system property that causes a particular simulation run to exhibit correct behaviour. More generally it is interesting to identify why each behaviour type occurs, though this is not necessary to use the architecture. In chapter 4 the parameters that have most impact on behaviour are identified as the internal weights, external inhibition and the connectivity. Together, these parameters control the amount of activation within the layers and the amount of inhibition needed to suppress them. The conclusion from this is that correct behaviour is caused by balancing the activation with the inhibition used to shut off the layer. This is not the only requirement, because incorrect behaviour can be seen in some simulations even when these parameters are balanced. The only difference between the simulation runs is the randomly allocated connections within each of the layers. This means that the particular connections between neurons have an impact on the behaviour of the system.

The analysis in this section is therefore concerned with the layer connectivity, and attempts to find a general property of this connectivity that will indicate the probability of a run being correct. For a particular parameter set the 'connectivity' is fixed, so all of the neurons have the same number of outgoing connections. This is not true of the incoming connections to each neuron, because for each connection the target neuron is chosen randomly from all of the neurons in the layer. This means that some neurons have more incoming connections than others. This will give them different incoming weight values, even when randomised weights are not used. For this reason the weights and number of connections that go to each neuron are examined in the following sections. Additionally, as self-connections are able to cause a neuron to keep itself spiking they are considered as a possible contributor to the behaviour.

5.2.1 Self Connections

The number of self connections that each neuron has is important, because the more self connections there are in the network, the more extreme the spiking could become. In a situation where a neuron only requires one input spike to produce an output spike a neuron will be able to keep itself firing indefinitely. This may not be as extreme using the thresholds chosen for the neurons in the simulations presented here, because each neuron requires multiple spikes to produce an output. From testing using these parameters it is found that six spikes of weight 1 are required to produce an output. If a neuron has multiple self connections then it will be more likely to maintain spiking.

To identify what impact the number of self connections has on the behaviour, three different examples of each type of behaviour are compared. This is shown in figure 5.1. From these examples it can be seen that there is no trend for the number of self connections per neuron with behaviour type. There are some fluctuations between runs, but all show a mean of close to 1 connection per neuron. The median value of all three runs for each behaviour type is 1 self connection per neuron, for all of the layers.

The number of self connections that each neuron has is therefore not a measure that can be used to determine the type of behaviour that a particular configuration of connections will exhibit.

5.2.2 Input Weights

As the target neurons are chosen from the neurons in the layer with equal probability, not only do different neurons have different numbers of connections they have

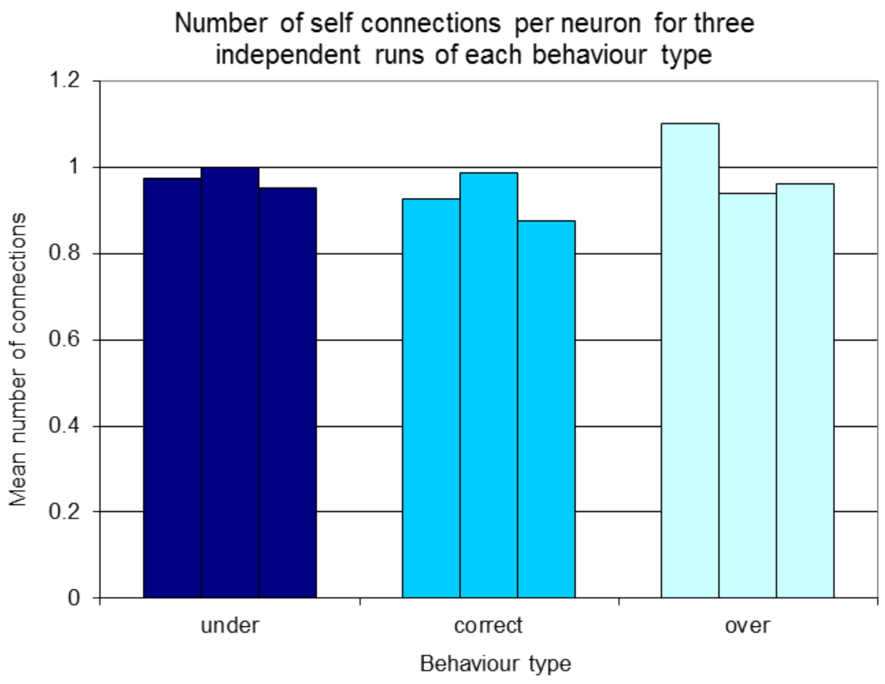


Figure 5.1: The mean number of self connections per neuron for each type of behaviour. There are three different runs of each behaviour type.

different total weight values on their inputs. The larger the weight values are the more quickly a neuron can be influenced, because integrate and fire neurons follow Pieron's Law (introduced in section 4.8). In the examples where randomised weight values are used the sum of inputs is continuous, but in the cases where a fixed weight is used the sums are discrete values (multiples of the fixed weight). The results presented here (figure 5.2) use randomised weight values, because it is easier to compare graphs where the values are spread out rather than clustered about the same points.

To identify the difference in connection weight for the three types of behaviour an example of each different behaviour type is examined. The sum of the input weights for each neuron is made up of both excitatory and inhibitory weights. The total of both values are plotted as coordinates for every neuron in the first layer in figure 5.2. The three behaviour types are represented by the different types of point. The excitatory and inhibitory values could be summed to give an overall weight value, but this would unnecessarily remove information. The location of the three groups of points on the graph, if they are shown to be separable, will allow classification of the behaviour type using the weight values.

Dale's Principle is not considered for these tests, because it only restricts the type of the outgoing connections from each neuron. These tests consider only the incoming connections to the neurons which can be either excitatory or inhibitory even when Dale's Principle is followed.

The different types of behaviour could be caused by the connections in any of the layers individually, or by a combination of many or all of the layers. For this reason all of the layers are considered separately. Layer 1 is shown in figure 5.2 and the rest of the layers are given in the appendix (figures A.3 and A.4). The example presented here uses 80 neurons per layer.

The graphs show that there is a great deal of overlap between in the distribution of points for the three different behaviour types in all of the layers. As there is so much overlap in all layers, it is not possible to conclusively say that if the weights lie in a particular region then the behaviour will be of a certain type. However, there is a pattern in the outlying points on the graph. The outlying points belong to the incorrect types of behaviour. The three points that have an inhibitory sum of less than -140 all belong to the over inhibited simulation run, and three of the four points with an excitatory sum of more than 140 belong to under inhibited (the fourth point is over inhibited).

To examine the distribution of the points for each of the runs the standard deviation can be considered. Different measures of spread could be used, for example the range of the data points. In this case the standard deviation is chosen because it reflects the distribution of all of the points in relation to the mean. This is considered useful because it is believed that a system with extreme values, further from the mean, will have a larger influence on system behaviour. Systems with extreme values should therefore be more likely to show incorrect behaviour and conversely systems with values close to the mean would be more likely to exhibit correct behaviour. This is suggested by the outlying data points on the graph. If there are a number of points that lie further away from the mean then their influence should be larger than a single point, and therefore there should be more chance of incorrect behaviour. The standard deviation will reflect the number of extreme values, but the range will only indicate the magnitude of the most extreme value. The range would be preferable in the case of a single outlier, but the standard deviation can reflect presence of such an outlier and incorporates the values of multiple outliers. So while other measures of spread are possible the standard deviation is considered the most suitable choice for this analysis.

The standard deviation for correct behaviour is 15.8 for the inhibitory sum and 14.8 for the excitatory sum. The standard deviation for over inhibited is 18.4 for inhibitory and 18.1 for excitatory, and for under inhibited 19.7 for inhibitory and 17.6 for excitatory. This shows the distribution of points tends to be closer to the mean when the behaviour is correct than when it is incorrect.

The statistical significance of these results can be assessed using the F-test. This test is used to compare two sets of data and to give a level of confidence whether they come from normal distributions that have the same variance. Further details of the test used can be found in the appendix section A.4. The results of applying this test show that for layer 1 the difference in the variance (and therefore standard deviation) of the correct behaviour and both incorrect types is statistically significant with 90% confidence. The difference for all of the other layers is not statistically significant. The p values for all of the layers are provided in the appendix in table A.1.

This suggests that the data points having a smaller standard deviation, which means fewer neurons have higher than average input weights, should improve the chances of correct behaviour. The values of standard

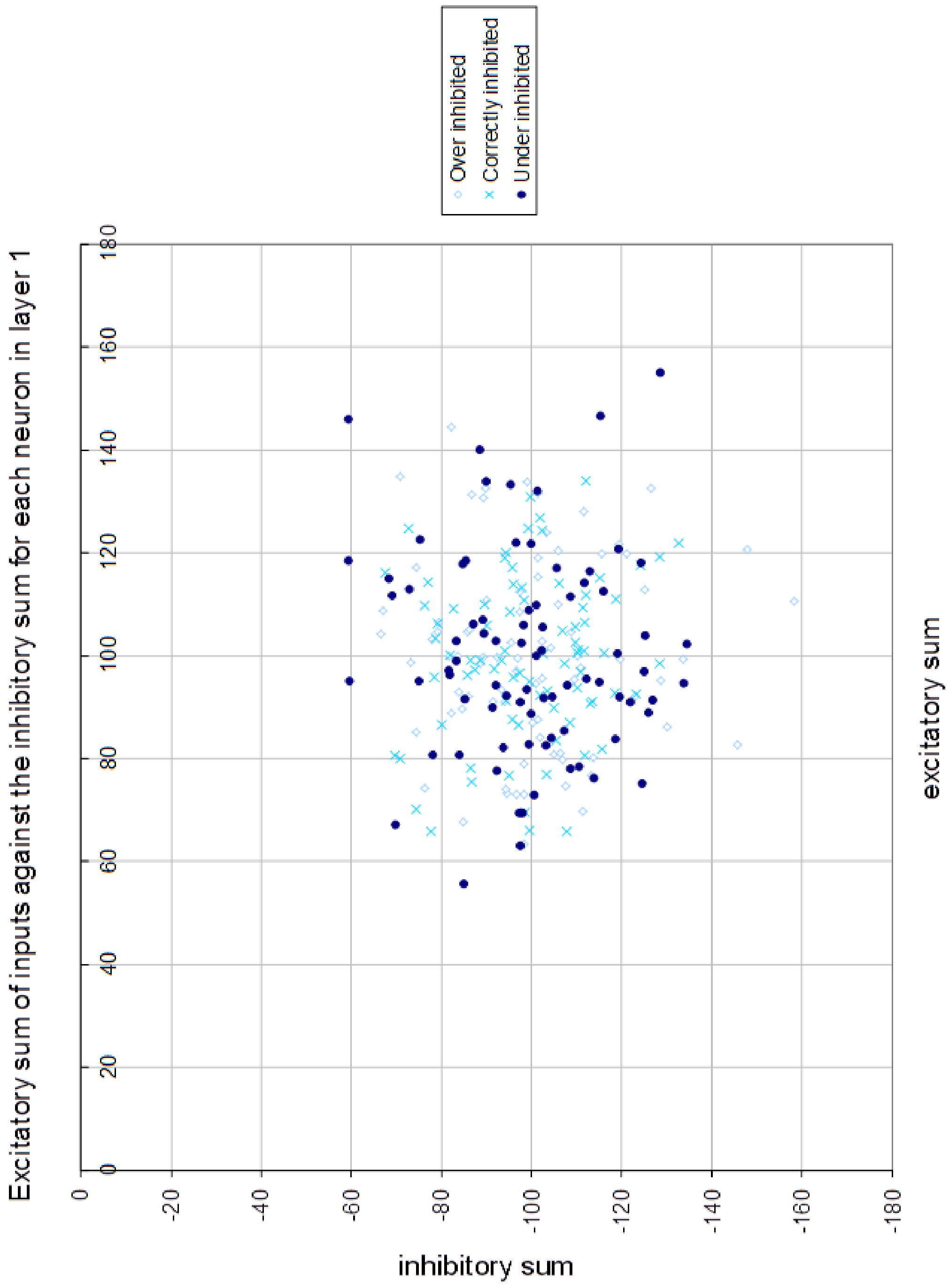


Figure 5.2: The sum of the excitatory weights plotted against the sum of inhibitory weights for the incoming connections of each neuron. One point on the graph represents one neuron in the network. The different types of point represent the type of behaviour seen with those weight values

deviation for all layers and all three behaviour types can be found in the appendix in table A.2.

Although the runs in figure 5.2 are single examples of each behaviour type, the differences in standard deviation suggest that it is not the average weight values that are important, but the distribution of the values. This suggests it is unlikely that having a number of neurons with large inhibitory weights and others with large excitatory weights could average out each other's influence and result in correct behaviour. The results suggest that it is possible for just two or three of the neurons with large weight values to adjust the system behaviour.

The standard deviation of the weights cannot be used to guarantee correct behaviour, instead it indicates the probability of a set of weights producing correct behaviour. This can be demonstrated by considering the following situations.

It is possible that a neuron with a large weight on its input connections may not influence the network much. This could be because it only has low weight values on its output connections or because it forms an island with other inactive neurons. As only some of the neurons in the first layer receive the system input not all of the neurons are stimulated, if some form an independent loop then none of these neurons will fire. Their weights could be very high and skew the standard deviation, while having no impact on the behaviour. This is the most extreme example, but there may be other cases where high weight values do not cause incorrect behaviour for example because they are connected to neurons with a low firing rate.

It is possible to imagine particular sets of connections that, even without extreme weight values, may cause incorrect behaviour. For example a chain of excitatory connections that are slightly higher than the average weight value may be enough to cause enough excitation in layer *n* to over inhibit layer *n-1*.

In the other layers there is not a distinct difference in standard deviation. The mean standard deviation over all of the layers for correct is 17.5, for over 18.0 and for under 18.6, but this is likely to be due to the large difference in the first layer. Apart from layer 1, only layer 4 has both excitatory and inhibitory standard deviations smaller for the correct run. **This suggests that with only one layer having a high standard deviation it is possible to cause incorrect behaviour.** This is reasonable, because as the external excitatory connections are only from one layer to the next, the data must pass through each layer. If only one of the layers provides too much or too little inhibition then it can prevent the data reaching the next stage.

5.2.3 The number of input connections per neuron

The previous section considers the impact of the weights on the inputs to each of the neurons on the behaviour of the system. Having some neurons with higher total input weights is thought to increase the chance of the run exhibiting incorrect behaviour. The larger weight values could be due to higher randomly chosen weight values on the connections, or because of a larger number of inputs to the neuron. To investigate this the number of inputs that each of the neurons receive is examined here. As with section 5.2.2 there is one run of each behaviour type for a network with layers of size 80. The weights in this example are fixed at plus or minus 2.5

to help identify whether it is the number of connections rather than the individual weight values.

Figure 5.3 shows the number of excitatory input connections that each neuron has, plotted against the number of inhibitory input connections. In this case the standard deviations of the three different behaviour types do not show the same pattern as in figure 5.2. The standard deviation of the correct run is not significantly less than that of the under or over run for any of the layers for excitation, and only one of the layers for inhibition. The p values are given in table A.3 in the appendix. **This suggests that it is extreme weight values on the connections that cause the variation found in section 5.2.2, not the number of connections.**

When fixed weight values are used (for example ± 2.5) the weight value and the number of connections are directly correlated. This means that there should be some information in the number of connections that helps to determine the behaviour. Therefore an alternative consideration is the overall weight value (number of excitatory compared to the number of inhibitory inputs). This represents how well the data is clustered about the diagonal $x = y$ in figure 5.3. The data for layer 1 is shown in figure 5.4, the standard deviation of the correct run is lower than for the two incorrect behaviour types, although the difference is not found to be statistically significant (using the F-test). The other layers also have no statistical difference between the different standard deviations.

The same process has been repeated for layer sizes of 70 and 60 neurons to test different examples of each behaviour type. It is found that for the examples of behaviour with 70 neurons there is no significant difference on any of the layers. With 60 neurons there is a significant difference on layer 1. The p values are given in table A.4 in the appendix.

These results suggest that it is not possible to use the number of input connections to each neuron to determine the overall behaviour of the system. This is reasonable, because as described in section 5.2.2 there it is possible to invent examples that have a high standard deviation and exhibit correct behaviour or a low standard deviation and exhibit incorrect behaviour. When the total sum of the input weights on each neuron are considered the standard deviations vary for correct and incorrect runs, sometimes significantly. **The results do suggest that the balance of excitatory and inhibitory inputs that each neuron has can sometimes be used to predict behaviour. Using a low standard deviation is a possible way to improve the likelihood of having correct behaviour.** This goes some way towards explaining the behaviour when fixed weights are used.

5.3 Identifying Inputs from Layer Activation

When the Neural Pipeline architecture is presented with a sequence of different inputs to process, it is important that it is able to distinguish between them. The system input is only applied to layer 1 so if two inputs result in patterns of activation that quickly degenerate to a state that cannot be told apart in layer 1, then it is unlikely that the later layers will be able to identify them as separate inputs. In [56] Rochel and Cohen investigate the preservation of identifiable patterns over time using

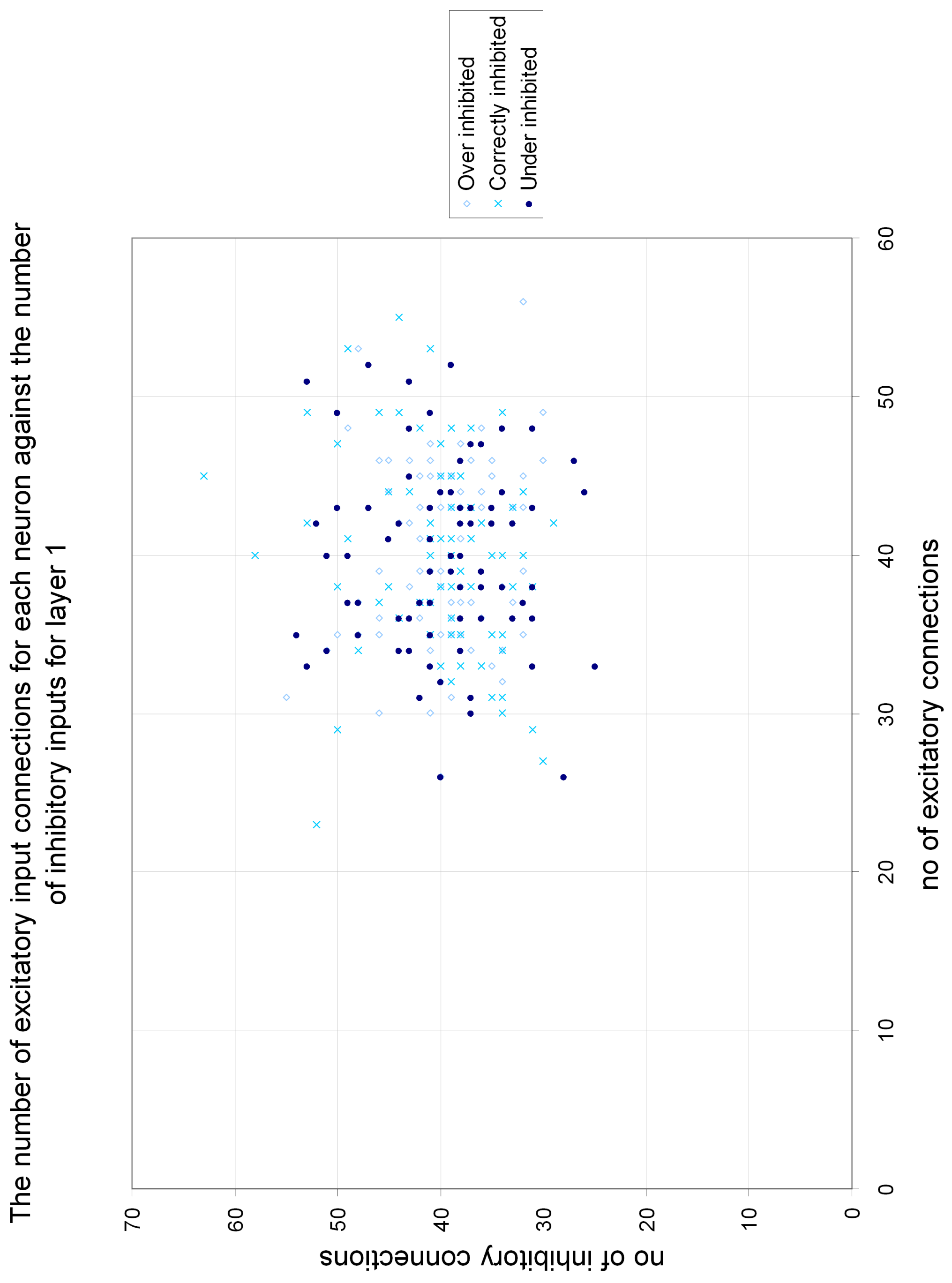


Figure 5.3: The number of internal connections that provide an input to each of the neurons in layer 1. The number of excitatory connections is plotted against the number of inhibitory connections. Each point represents the values for one neuron in the layer. The type of point (cross, circle or diamond) shows the behaviour type that is seen with these values.

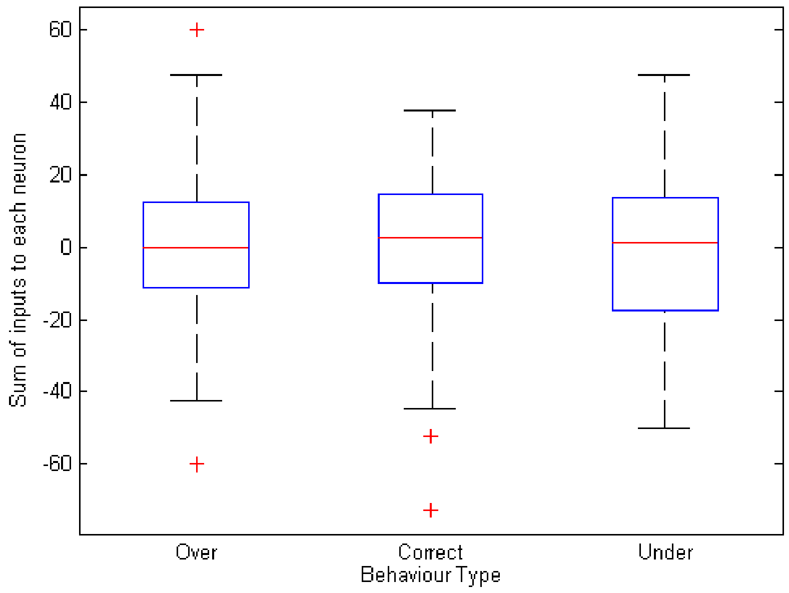


Figure 5.4: The sum of weights on the internal input connections to each of the neurons within layer 1, for an example simulation run of each behaviour type.

population coding. They use a randomly connected single layer network, similar in structure to one layer of a Neural Pipeline.

In order to measure how distinguishable the patterns of activation in the network are, Rochel and Cohen propose a distance metric. The metric compares the activity of each neuron in the network for two distinct inputs. This metric is found to be a mechanism for telling the duration that real inputs that are presented to a Neural Pipeline remain distinguishable. A variation on the method used in [56] is used to do this. Their original method must be introduced initially to explain the variation.

To consider how the distinguishability changes over time the metric is applied to windows of the simulation, rather than the entire simulation. Two different ways to choose the window are used, the first is simply to split the simulation into windows of time. If the spikes are not regularly spaced through the simulation, then windows of a certain number of spikes are used instead. In this case the window over which the comparison is made is defined by the time taken to accumulate n spikes in the network. In the first case time in ms is used for the x axis of any graphs produced using this metric and in the second case it is the cumulative total number of spikes.

The distance metric used to compare the patterns is given in equation 5.1. Here A and B are different input stimuli, where A corresponds to an input presented to a particular set of neurons and B an input presented to a different set. $N_{A,i}(x)$ is the total number of spikes that neuron i produces in a given window x in response to stimulus A. $N_{B,i}(x)$ is corresponding value for input B. In each window any of the neurons in the network can either fire or not fire. The neurons that fire make a contribution to the distance. To allow comparison the value is normalised to a value

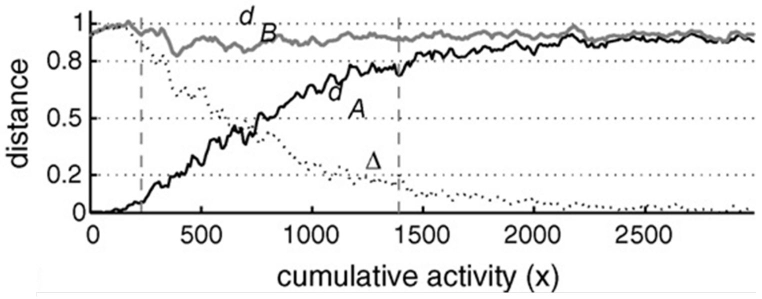


Figure 5.5: Graph from [56] showing the average value of the distance metric for two different inputs A and B. Line d_A represents the distance between the initial response to input A and the current state of the network after A is applied. Line d_B represents the distance between the initial response to input A and the current state of the network after B is applied. The x axis is 'cumulative activity' because it uses a number of spikes per window, rather than an absolute time. The Δ line is the difference between d_A and d_B. (The vertical lines represent the readout window defined by Rochel and Cohen, this definition is specific to their work, so these lines are not considered further.)

between 0 and 1, where 0 means that the two windows contain exactly the same number of spikes from each neuron, so there is no distance between them.

$$\sum_i |N_{A,i}(x) - N_{B,i}(x)| \tag{5.1}$$

To compare the behaviour the spike response from the first window of stimulus A is used as a reference. Using the distance metric, this is compared to the response from all other windows of stimulus A. It is also compared to all windows of stimulus B. A sliding window is used to produce the graph shown in figure 5.5

The line d_A represents the distance between the response from A and the reference. The line d_B is the distance between the response from stimulus B and the reference. This means that d_A always starts at 0, because for the first window it is compared to itself. It then increases as the pattern of activity changes and becomes more different. The graph shows that over time the two unique inputs become indistinguishable. The aspects of interest are the initial distance between the signals and the duration (in cumulative spikes or ms) that the two inputs remain distinguishable.

The results of using the distance metric in equation 5.1 for a 5 layer Neural Pipeline are shown in figure 5.6. The parameters used can be found in the appendix in table A.5.

The simulation is averaged over 10 different inputs each with 10 different sets of internal connections. Each run had different internal connections, chosen at random, so the number of spikes produced in each individual simulation varies. Since the metric is based on observed spikes, the comparison can only be made when the neurons are active. This means that the comparison stops after the shortest simulation run. This is the reason for the difference in length seen between runs for input A and input B.

The window size used for the simulation is 50 spikes to follow the requirement in [56] that the size is chosen 'to be sufficiently large to ensure our distance metric is viable yet sufficiently small that all or nearly all neurons have fired at most once'. The importance of the window size is investigated in the appendix section A.4.1.

When comparing the results of applying the metric to a Neural Pipeline (figure 5.6) with the results on a single layer from [56] (figure 5.5) it can be seen that a similar result is found for each layer. Both inputs are well separated at the beginning of the simulation but then degenerate to a similar state. The main difference is that layers 1 to 4 are terminated by the external inhibition of the system cutting off the activity in the layer more quickly in these layers than the last.

5.3.1 Comparing Specific Inputs

In the previous section the tests used a series of randomly chosen inputs to test the average distance. The following tests use specific shape inputs, that are used in the following chapter to test how the architecture can be trained on simple images. These tests show how distinct the shapes are as they propagate through the architecture. The tests use the parameters from those experiments (given in table A.7 in the appendix). The examples presented here measure the distance between the three shapes shown in figure 5.7. These examples are chosen because square and plus have similar shapes, with the same number of active inputs, but cross has fewer active inputs and a more distinct shape.

When these specific inputs are used the cumulative spike total is not a sensible way of examining the data. This is because the number of spikes in the layers is small compared to the results in the previous section. This means that any individual neuron can make a large difference to the distance value. This is explained further in the appendix section A.4.1. To avoid this sensitivity time windows are used instead.

An example of the response for square compared to cross is given in figure 5.8 (a) and square compared to plus in (b). A window size of 30ms is used for these tests. In both cases the different inputs start off with different values and then converge towards the same value. Compared to the example in [56] the simulation takes fewer milliseconds to complete, so when the two values converge it is because there are no longer any spikes in the layer so they automatically have the same value.

Square and plus are more similar to one another than square and cross are, as can be seen by looking at the number of active neurons that the shapes share in figure 6.5. When the square and plus graphs are compared to the graphs for square and cross in figure 5.8, by the time the signal reaches layer 3 the distance between them is much smaller. This is expected because the inputs are more similar. In [56] the cut off used for when a signal is considered distinguishable is 20%, but the difference in this case is less than 5%. It is shown in section 6.3.2 that the system is able to learn all of the shapes on all three of the layers. This means that either 5% is a sufficient difference to allow the system to learn correctly, or the measure used does not convey all of the information that is present in the signal.

The metric compares the current window of one input to the first window of the other input, but in training the architecture the same two windows are compared

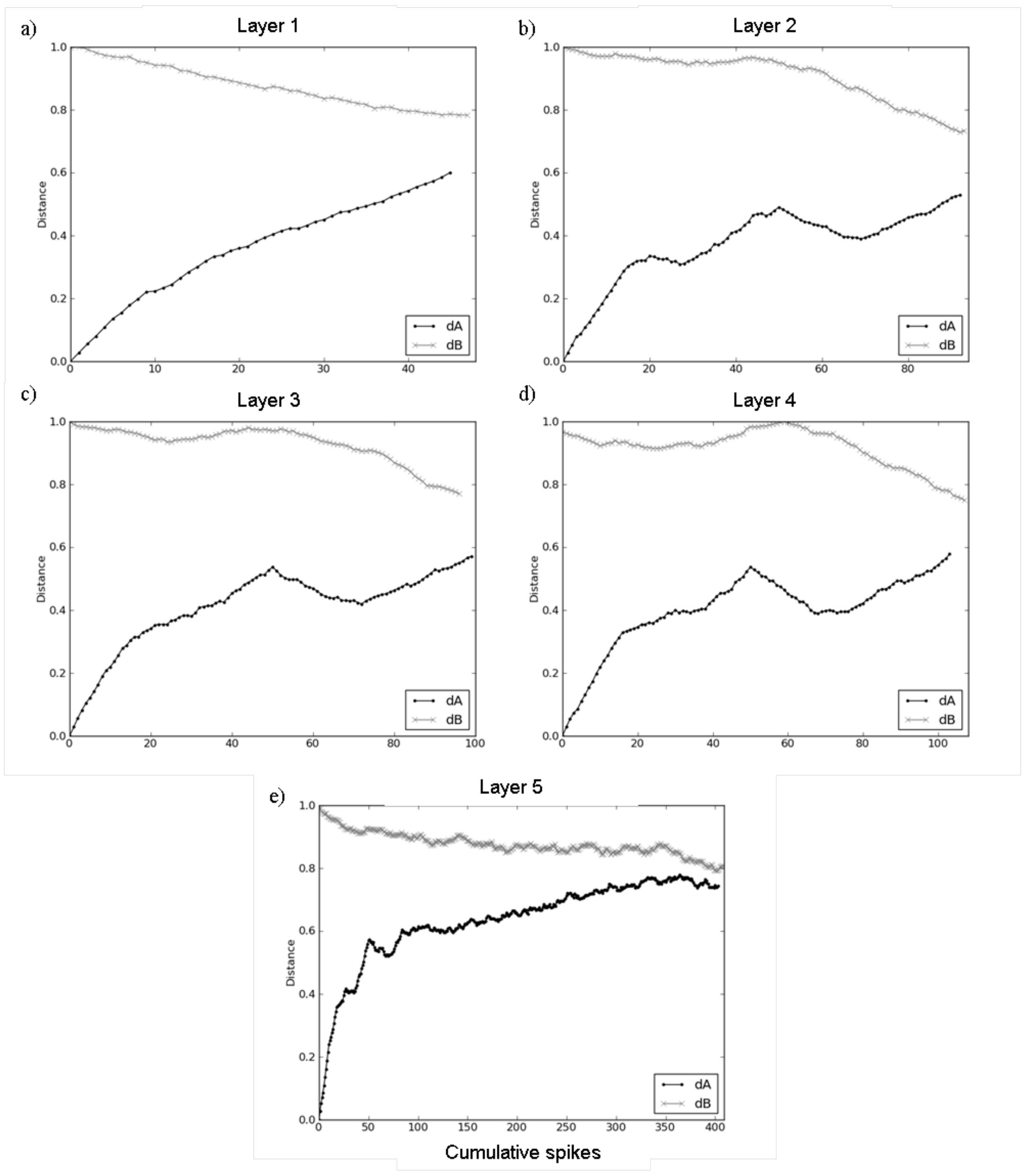


Figure 5.6: The distance metric plotted for each layer in a Neural Pipeline. The graphs averaged over 10 different input stimuli and 10 different internal setups for each stimulus.

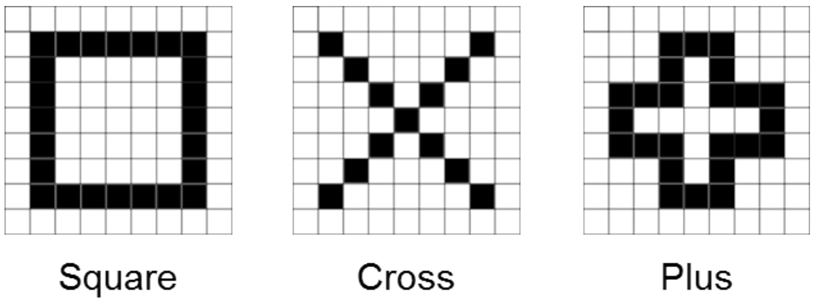


Figure 5.7: The three input shapes used to test the distinguishability of example inputs on each layer of the architecture.

directly. For this reason an alternative version of the metric is considered, which compares input B to input A at each time window. The results are shown in figure 5.8. The comparison of A is now always 0, because it is compared to itself. It is the comparison of A with B that is interesting. This now shows that the signals are very different (have a large distance) until around 10ms. When layer 3 is compared for (a) square and cross and (b) square and plus, it can be seen that there is a larger distance in graph (a) than (b). **This version of the metric is considered to be more useful when comparing the inputs for training in a Neural Pipeline, because it is easier to compare how different two inputs are after a duration. This is useful when deciding which window to use for training, and to tell which of the inputs will be more distinct.**

5.4 Discussion

Two different ways of analysing the system have been presented. The first considers the underlying cause of three different types of behaviour. The second considers how the system is able to distinguish two different inputs as they pass through each of the layers of the architecture.

The behaviour of the system depends on the random nature of the connection choices and of the connection weight values. Three different properties have been examined to identify how they contribute to the behaviour. These are the number of self connections, the total input weight to each neuron and the number of inputs that each neuron has.

It is found that the average number of self connections a neuron does not control the system behaviour. The three different behaviour types were all found to have an average of 1 self connection per neuron. This means that it is not possible to identify whether a run will be correct by observing the number of self connections in the network.

The standard deviation of the sum of input weights of the neurons within a layer is a contributing factor in the behaviour of the system. The value can be used to give a prediction of the behaviour type, because large values of standard deviation mean that the system is more likely to behave incorrectly. This is identified using an exam-

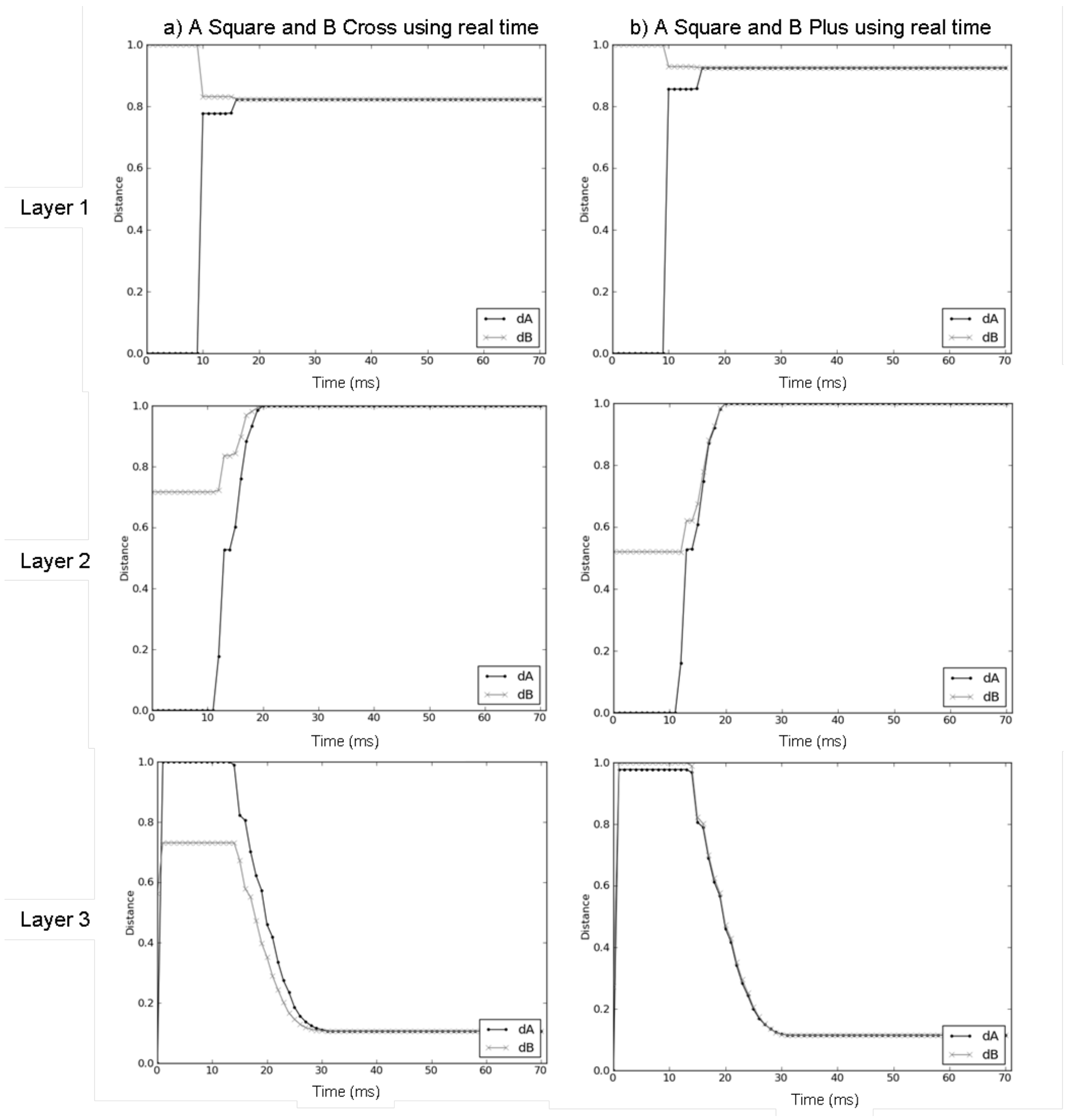


Figure 5.8: (a) The distance between square and cross, using the distance metric with real time and (b) the distance between square and plus. Window size of 30ms.

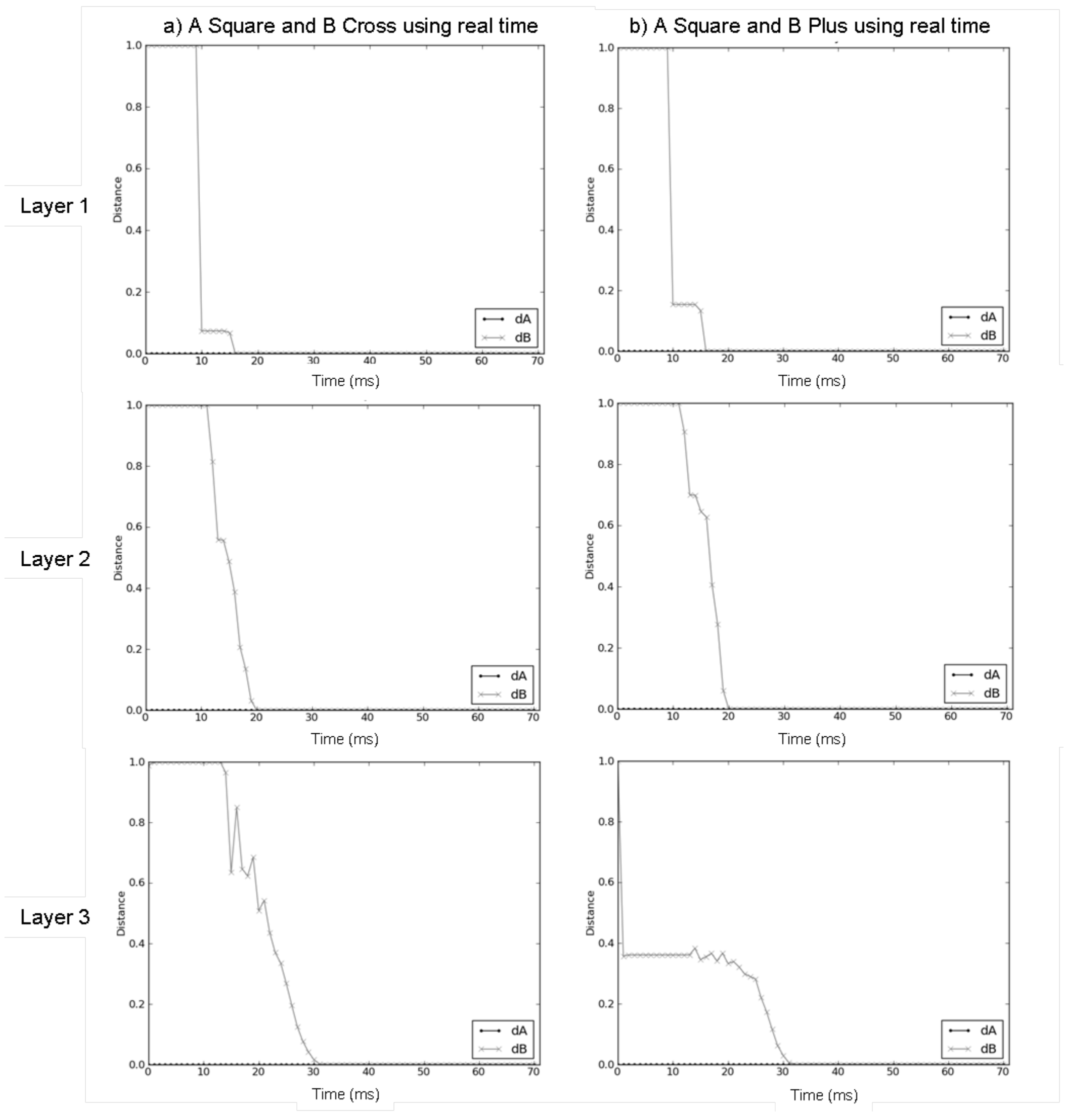


Figure 5.9: A revised version of the metric that compares each window with the corresponding window from the other signal.

ple of each type of behaviour and finding that the correct and incorrect behaviours have a statistically significant probability of coming from different distributions. It is believed that as the neurons with larger weight values have a higher influence on the network, they will be more likely to contribute to high or low excitation and to cause incorrect behaviour.

Restricting the standard deviation, either by manipulating the choice of connections when they are generated or by selecting only certain networks from all of those generated, should improve the chance that the system will behave correctly. The stochastic nature of the connections means that even with these constraints it is not possible to stop incorrect behaviour completely, just to minimise the chance of choosing connections that will cause it. The example presented demonstrates that prediction is possible, but the actual values of standard deviation are still to be determined. They should be addressed in future work as described in the following section. In the example examined only the first layer showed a significant difference in standard deviation between the three types of behaviour. The interaction of the layers has not been identified in this work, this is also recommended for future work.

The number of connections contributes to the weight value, because the higher the number of connections the higher the weight can be. The standard deviation of the number of connections is not shown to be a mechanism for predicting the behaviour type. Instead the balance between inhibitory and excitatory connections is found to give some indication of behaviour type, although only one of the presented examples has a significant difference in standard deviation. This means that the standard deviation of the excitatory to inhibitory connection balance cannot be used as a reliable measure of behaviour type.

To identify the preservation of information through the system a distance metric proposed by Rochel and Cohen [56] is used. It is found that a variation of the comparison method can be used to show how distinct two different inputs are at any time in the simulation. Comparing the graphs allows for a possible method of selecting which window to train the system on, training and window selection is described further in section the learning tasks carried out in chapter 6. The metric can also give an indication of how similar the different inputs are when compared to one another, this gives an indication of how difficult the training will be. It is a possible way of choosing how to split the inputs between the different layers when learning.

5.4.1 Future Work

The analysis carried out provides an initial insight into the possible causes of correct and incorrect behaviour, but further tests would provide a wider understanding.

The graphs shown in section 5.2.2 suggest that layers with a high standard deviation of excitatory and inhibitory input sums are more likely to produce incorrect behaviour. It is suggested from this that a value of standard deviation could be found that will provide correct behaviour with a probability of for example 95%. This provides a mechanism for replacing sets of connections that are likely to cause incorrect behaviour. This may be useful when choosing a set of connections for the architecture, so is a possibility for further work. It may however be more beneficial to investigate other aspects because it is easy to generate a new set of connections and test whether it behaves correctly under the required conditions.

The results from the sum of input weights (section 5.2.2) also suggest that the layers may have a different impact on the system. Two alternative hypotheses are proposed to suggest which of the layers may have more or less chance of altering behaviour.

The first hypothesis is that the middle layers may be more likely to have an impact on the behaviour than the end layers because they have both feedforward and feedback connections, so can influence two other layers. This does not mean that layers 1 and 5 cannot cause incorrect behaviour, just that they can only influence one other layer. The example correct run shown here has a higher value of standard deviation on layer 5 than either the over or under inhibited run. This may be because layer 5 has a lower impact, or it may be because any of the layers can have a higher standard deviation but still be correct because of their particular connections.

The second hypothesis is that earlier layers can have more of an influence on behaviour because the input is only provided to the first layer of the system. If layer 1 has very extreme values then the effects of this may be seen throughout, but if layer 5 has extreme values there are no further layers to pass the signal to.

The two hypotheses cannot both be true, so they should be tested to identify if either one is true. Knowing this will mean that it will be easier to influence the system to have correct behaviour by concentrating on improving the parameters of those layers that have the most impact.

Another consideration is that the layers may interact to cause incorrect behaviour. One layer that has a high standard deviation, followed by another may cause more extreme levels of activity. For example if a layer that has more extreme negative input connections is followed by a layer with more positive input connections, it may increase the chance of over inhibited behaviour. This is because the second layer can more strongly inhibit the first layer and any spikes in the first layer are likely to cause more inhibition. It is also possible that if two consecutive layers have larger standard deviations then they could improve the overall behaviour, because one could balance out the other.

Another possibility is that the direction of skew of the data could be suggestive of type of behaviour. Anecdotally, from figure 5.2 the over inhibited run has more neurons with a high inhibitory sum and the under inhibited run has more with a high excitatory sum. It may provide an interesting avenue for future work, but it is not essential because it is not important which of the incorrect behaviours is exhibited, just that the run is not correct.

When the architecture is tested on larger and more complex sets of inputs the distance metric can be used to give an indication of how difficult the inputs will be to learn and their separation within the architecture. This should enable more careful choice of which input should be learnt by which layer in the system. This is a consideration for future work.

5.4.2 Summary

A Neural Pipeline has been analysed in two different ways, to identify the cause of the different types of behaviour and to identify the distinguishability of inputs. The behaviour of the architecture is influenced by the specific connectivity of each of the layers. A general property that contributes to the behaviour is the sum of weights on the input to each of the neurons within a layer. The chance of a correct run can be increased by having a low standard deviation of input weight. The distinguishability of different inputs, as they pass through each layer, can be identified using a distance metric.

The following chapter presents empirical tests that illustrate the preservation of data through the layers of the architecture. It demonstrates how different features of an input can be extracted using different layers of the architecture and examines the impact of noise on the system.

Chapter 6

Neural Pipeline as a Reservoir Computer

6.1 Introduction

The Neural Pipeline architecture has been developed to investigate the hypothesis that coordinating the activity of firing neurons represents a method of controlling behaviour in spiking neural network memory, in the context of computational applications. Chapters 3 and 4 have demonstrated that the architecture can control the flow of activity from one layer to the next. For the application of the architecture to pattern recognition, the Neural Pipeline must be able to learn to process specific inputs and respond accordingly with an output. Different shape inputs are presented to neurons in the first layer in the system. The activity then propagates through the layers in turn. Each layer has a separate set of readout neurons attached to it, these are neurons that are trained to display a relevant output when they detect a particular pattern within the layer. This is achieved by viewing each layer as an individual Liquid State Machine (LSM). This approach has been chosen because it captures a way of introducing learning within a randomly connected architecture. Each layer of a Neural Pipeline is a randomly connected network so with the addition of readout neurons a layer can be used as a LSM. As the training takes place on the connections to the readout neurons the process of training does not influence the layers themselves. This means that the results seen in the earlier chapters still hold if the LSM method is used to train the system. LSM have been successfully applied to similar tasks [49, 39, 70, 71, 48, 12] and are biologically plausible because they can respond to time-varying inputs in real time [49]. As the Neural Pipeline is biologically inspired for computation of time varying inputs the LSM is a suitable choice. To test the system as a multi-layer LSM, three hypotheses are proposed and tested in the following experiments.

The first hypothesis:

> ‘Data presented to the input layer of the pipeline can be recognised at every layer’

is tested using the shape experiment (section 6.3). This experiment considers whether the multi-layer system is able to identify a set of shapes. The input is presented only to the first layer and the shapes must be identified by each of the layers. The layers are considered to successfully identify the input if the readout neuron trained to recognise that input fires most strongly. This experiment tests whether information is still identifiable after it has passed through the system.

The second hypothesis is based on the requirement of the system to abstract different properties from a given input. It is hypothesised that:

> 'Different layers can recognise different properties of the same input'.

Example properties could be the size, shape, colour, position or rotation of an image. The second experiment is used to determine whether it is possible to filter out different properties of the input in different layers of the system.

The third hypothesis relates to the introduction of imperfect inputs to the system.

> 'Noisy system inputs should cause a system trained on perfect inputs to fail gradually, depending on the magnitude of the noise.'

The third experiment introduces different types of noise into the inputs to identify the robustness of the system, to see how quickly and to what extent the system fails.

The design decisions common to all three experiments are outlined and justified in the following section. Many of the parameter choices for the Neural Pipeline are influenced by the paper that introduced the concept of the Liquid State Machine [49]. The experiments from the paper are not replicated, because here the application is image recognition and none of the three learning experiments in [49] perform this task. Instead, a set of experiments based on image recognition have been carried out. They use an input grid, drawing inspiration from existing work on image recognition in LSM [39, 70, 71, 48, 12].

This chapter demonstrates how the Neural Pipeline can be used for learning and how the different layers can be used to perform different tasks. The examples given here are considered to be a proof of concept, to show that the Neural Pipeline is a usable architecture. The future work section outlines how the architecture can be developed further for use on more realistic tasks.

6.2 Experimental Setup

The application chosen to test the Neural Pipeline is image recognition, because it is a task well suited to neural network memories. To provide input for these experiments a grid of 81 pixels is used, arranged into a 9 by 9 square. Each pixel in the grid provides the input to one of the neurons in the first layer of the Neural Pipeline. Black pixels provide a spiking input to the neuron that they are connected to and white ones remain silent.

Each pixel stimulates a different neuron in the first layer so that the data is not compressed. This means that there are a minimum of 81 neurons in the first layer. If the same neuron received inputs from multiple pixels the layer could not identify which of the inputs was active, only the average behaviour. A 9 by 9 grid is used because it balances the need to have a grid large enough to convey different images and the need to reduce computational expense. Larger grids mean larger layer sizes, which produce more spikes and increase the simulation time. It is the smallest grid that could be used for the positional experiment (section 6.4), with a distinct shape in each of the sections. A simple black and white grid was chosen to produce a simple image recognition problem. The background colour (white) was chosen as the non-spiking colour to reduce the number of spikes and therefore improve simulation efficiency. It would be possible to extend the experiments by using a larger grid size or introducing different colours in the input. Each colour could be represented using a different firing rate.

Once the size of input grid is set at 81 neurons, the number of neurons per layer is based on this choice. As one neuron receives each input pixel there is a lower limit of 81 neurons in the first layer. The experiments carried out in section 4.6.5 suggest that a smaller input compared to the layer size can increase the chance of correct behaviour, so 100 neurons was chosen as a suitable layer size to compromise between this and keeping simulation time to a minimum. Testing 100 runs, with the parameters to be used for the experiments, all 100 had correct behaviour. This shows that 100 neurons is a large enough choice of layer. As with the standard Neural Pipeline architecture described in section 3.2 the same number of neurons were used in each layer. This does not compress the data to fewer spikes between stages, but this is a consideration for further work (see section 6.6.1).

In these experiments the input stimulus presentation duration is 10ms for the shape and positional experiments (sections 6.3 and 6.4) but varies in the input experiment (section 6.5). This differs from the examples in [49] where the input is provided for the entire run time. It is important to present and stop the stimulus for these experiments because the Neural Pipeline has been designed to be presented with a stream of inputs to process. It is possible to present a continuous input, but the architecture will ignore the input while it is processing. If a response is required from each of the inputs then a separated stream needs to be presented. This means that unlike the readout in [49] the readouts in the Neural Pipeline will only display the result for a short interval rather than the entire run time. This is more similar to the behaviour of the readouts in [48] where the input is a stream of data. In this case though the stimulus is presented and removed, rather than being continuous.

The learning algorithm used in the experiments outlined in sections 6.3 and 6.4 is the delta rule (see section 2.5.2 for a description). The delta rule was chosen rather than the p-delta rule [7] used in [49] because of the choice of a single readout per input rather than a large group of readout neurons. The p-delta rule is used for parallel perceptrons, but can also be used for groups of integrate and fire neurons [49]. The p-delta rule comprises two parts. The first is the standard delta rule, the second is a rule to determine which of the weights the delta rule should be applied to. In [7] they choose to apply the rule to all weights that are incorrect, but introduce a margin of error to stop weight values changing sign when they are close to zero. This margin of error is not necessary in the following experiments because they are trained without noise. For larger groups of readouts or noisy simulations the p-delta rule is preferable, but for these simulations the delta rule is sufficient. Alternative learning methods could be used to train the Neural Pipeline (for example those discussed in 2.5), but in this case the delta rule was chosen because it has been successfully used for LSM in [49] and [22].

The delta rule, when applied to a spiking network, takes the total number of spikes from each neuron and adjusts the weights by this sum multiplied by the

```
for all readout neurons do
  if $current\_readout\_output > required\_output$ then
    for all weights do
      $new\_weight = old\_weight - (learning\_rate \times neuron\_output)$
  else if $current\_readout\_output < required\_output$ then
    for all weights do
      $new\_weight = old\_weight + (learning\_rate \times neuron\_output)$
  else if $current\_readout\_output == required\_output$ then
    for all weights do
      $new\_weight = old\_weight$
```

Figure 6.1: Pseudocode for the Delta Rule.

learning weight. Therefore if the neuron does not spike no change will be made to the weight, otherwise the weight is adjusted proportionally with the number of spikes. It is increased if the output is too low, when the readout does not fire and is meant to, but decreased if the output is too high, when the readout fires when it should not. Pseudocode showing the delta rule when used for LSM is given in figure 6.1.

To use the delta rule in a LSM the internal state is used to train the system rather than the system input. As the state is dynamic a suitable way to identify a particular state is to split the activity into time windows and record how frequently each neuron spikes in a given time window. Training on a time window means that the readout neurons will identify the pattern as it passes through that particular time.

The examples in [49] are not split into windows for training, so it is assumed that they use the entire simulation duration to train the system. Even if the training is not broken into windows it is effectively a window of the same size as the training time. If the simulation is run for longer than the training time then this is equivalent to breaking up the signal into windows for training. The Neural Pipeline architecture is designed to be run for longer than the training simulation time, because a series of inputs is to be presented to it, therefore training based on time windows is a sensible choice.

Throughout this series of experiments the window size is fixed at 5ms. This duration was chosen so that each neuron tends to only spike one or two times within a window. The reason for limiting spiking is to try to increase the likelihood of finding a unique window for each of the inputs. The choice of window size may influence learning, but as the experiments were designed to investigate other properties of the Neural Pipeline this parameter was fixed throughout the experiments.

The window selection method used for this set of experiments is to choose the first (chronological) set of windows which is unique and non-zero for each shape. All of the windows must be non-zero, because if there are no spikes it is not possible for the readout neuron to fire. In these experiments the same numbered time window (e.g. window number 5) is used for all of the shapes, so that the readouts take the same amount of time to respond for each of the shapes. Figure 6.7 on page 137 shows the results of a simulation with the resulting spike times displayed by splitting them into time windows. These windows were then used to train the system for the experiment. So for example in layer 1 the chosen window in this case is window 2, because it is the first where all shapes have a unique, non-zero response. In this instance it is the only choice of window that meets these criteria. This suggests the possibility of two unique shapes being presented, but no unique windows being available upon which to train the system. In this case the window size could be reduced to achieve a finer representation, and hopefully a set of unique windows. An example of this is given in figure 6.2. The raw spike time data is shown in (a), each neuron spikes twice over a 20ms period. When the data is split into two 10ms time windows in (c) both windows are identical, so there is no unique window on which to train the system. By halving the window size, as shown in (b), it is possible to produce three (non-zero) unique windows, any of which could be used to train the system. There are instances when this technique would not work, such as if none of the inputs produced any activity within the layers. In this case other parameters such as the neuron threshold or the magnitude of the input would need to be adjusted.

There are clearly alternative methods for choosing which set of windows to use, because there are a large number of windows. So for example the second set could be used rather than the first, and so on. It is possible to choose windows with the greatest possible separation between the shapes, using separation as defined by Maass et al in [49] or using the distance metric from section 5.3. In this case the first window was chosen in order to minimise the amount of preprocessing required before training. In order to find the windows with the greatest separation it is necessary to search through all of the windows. Even if it was determined that the most separated windows were likely to appear within a certain time, based on the 'distance metric' which suggests they should be early in the simulation (see section 5.3), it would still take longer to search through all of these windows than using the first set. The use of the first unique set of windows also serves to illustrate whether the Neural Pipeline is able to distinguish shapes even without idealised separation.

As in [49] the readout neurons are connected to all of the neurons in the LSM layer. There are no lateral connections between the readout neurons either intra-layer or inter-layer so that the output is based solely on the response from the liquid layer. If connections were added between readout neurons then additional parameter choices would have to be made and this could influence learning ability, so as a baseline these experiments have no such connections. Adding these connections is a possible extension of the work. The initial weights for the readout neurons are set to values chosen randomly between 0 and the internal excitatory weight value (in these examples 0.5). The choice of initial weights is not a parameter that has been chosen to be varied.

The initial values are scaled to be in the same region as the internal weight values because the internal weights influence how frequently the neurons spike. The weight change during learning is calculated using the number of spikes multiplied by the learning rate, so if this is very small compared to the initial values then learning will take a long time. Therefore keeping the initial weight values similar to the internal values should minimise excessively long learning times. The values are not

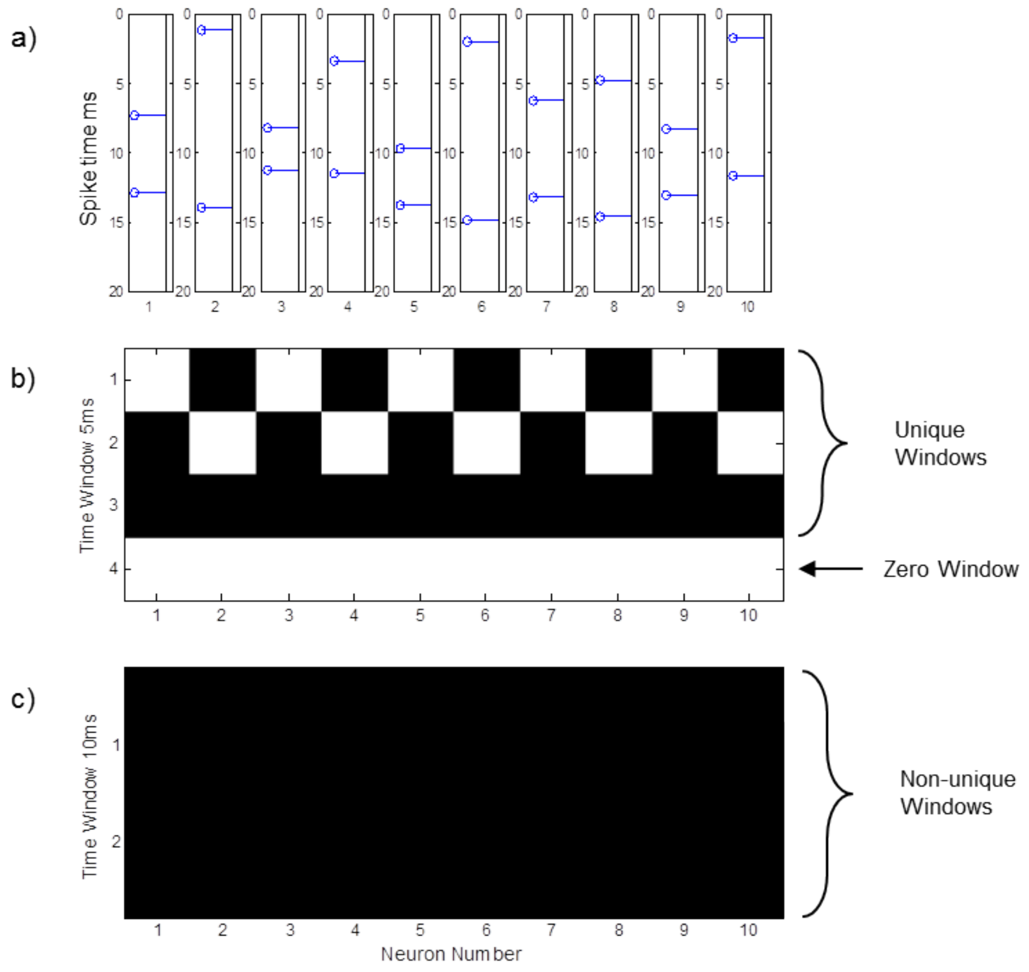


Figure 6.2: Example of the loss of unique windows when a larger window size is used. Example spike times produced by 10 different neurons are shown in graph (a). Graphs (b) and (c) both represent the same data by splitting it into different time windows. (a) has a window size of 5ms and (b) is twice as big with a size of 10ms. In (a) there are 3 unique (non-zero) windows, but there are none in (b). This illustrates how choosing a large grained window can lose information.

```
Input = First Input
while Any readout response is incorrect do
  Run simulation with current input
  Update weight values
  Check readout correctness for all inputs
  Next input
  if Input == Last Input + 1 then
    Input = First Input
```

Figure 6.3: Pseudocode for learning algorithm 1.

optimised, because the actual learning time is not investigated here. The problem of initial weights meaning that the learning will get stuck in local optima is overcome by rerunning the learning algorithm with different initial weights if no progress is being made.

The learning rate is the step size that the weight takes each time it is updated. The learning rate is set to 0.1 throughout the experiments. This value was chosen to be lower than the average initial weight value (with few spikes per window) but large enough so that it could change the sign of the initial weight in only a few learning steps. The choice of learning rate mainly alters the speed of convergence on a set of weights that provide the correct output. It is possible with too large a learning rate that the steps will be too large making it impossible to arrive at a correct set of weights. A description of the influence of the learning rate can be found in [50]. If the result is correct then only the time taken is changed, therefore the learning rate parameter is not investigated.

Two alternative learning algorithms were considered; the first loops through the individual inputs (e.g. each of the six shapes in the shape experiment section 6.3) and performs one update per shape until the output is correct. This algorithm is shown in pseudocode in figure 6.3. The second updates the weights for one input pattern until it is completely learnt then moves onto the next pattern, pseudocode is shown in figure 6.4. As learning a new pattern can unlearn an old one, this process must be repeated until all patterns are correctly learnt, rather than being able to exit after any one of the input patterns is correctly learnt. This is shown by the ‘For all inputs’ loop in the pseudocode for learning algorithm 2 (figure 6.4), and absence of the same loop in the pseudocode for learning algorithm 1 (figure 6.3). Both algorithms were shown to work for the shape example in section 6.3. It was determined that the first algorithm would be preferable because it makes incremental progress each time rather than taking steps which may be in the wrong direction before correcting them.

The number of readout neurons used was chosen to minimise computation and to be easy to observe. The lowest number of neurons that is easy to observe is one neuron for each pattern property. For the shape experiment (section 6.3) the patterns are shapes, so there is one readout neuron per shape. For the positional experiment (section 6.4) the properties are shape and position, so there is a readout for each shape and one for each position. The noiseless inputs and lack of noise internally mean that single readout neurons are acceptable, although a group of readout neurons

```
while Any readout response is incorrect do
  for all inputs do
    while any readout response is incorrect for this input do
      Run with current input
      Update weight values
      Check correctness for current input
  Check readout correctness for all inputs
```

Figure 6.4: Pseudocode for learning algorithm 2.

would make the system better able to generalise (this is discussed in the further work section 6.6.1). As there is no noise the training for the readout neurons was chosen to be fairly strict. The readouts may only spike for their corresponding pattern, they must not spike for any other pattern. They may spike any number of times for their own pattern. Alternative responses would be to spike with a certainty relating to the input, with larger numbers of spikes representing a higher certainty that the pattern matches what the readout is trained to recognise.

The first two experiments (section 6.3 and 6.4) have been designed to be noiseless. This is because the experiments are designed to test whether information is still identifiable through all layers of a Neural Pipeline, rather than the ability of the system to identify imperfect inputs. The third experiment (section 6.5) addresses this ability, once it has been shown that the information is available in all three layers.

Maass et al [49] show that recurrent connections are important in LSM. This influenced the decision to allow multiple connections between a pair of neurons and self connections for the Neural Pipeline architecture (as described in section 4.6.3). The choice of randomised connections uniformly chosen between all neurons in a layer should also provide recurrence. In [49] they adjust a parameter (λ) which controls the connectivity provides to find the best average correctness over 50 runs. The result matches that found in the tests on a Neural Pipeline, that a lower level of connectivity produces more correct behaviour as seen in section 4.6.3. In Maass' results there is a drop off seen at lower values, which is not found in the Neural Pipeline tests. A possibility is that it may occur lower than the lowest tested connectivity. This means that the connectivity value chosen for the Neural Pipeline, based on these tests, should be acceptable.

LSM liquid layers can be trained to improve the system performance, as seen in [51, 33, 15]. The layers in the Neural Pipeline have randomly chosen connections and are not trained. There are several reasons for this choice, not training the layers means that the data in the later layers is not altered. Training one layer may be detrimental to the following layers by reducing information content. Training is not necessary for the operation of the system, it can just offer an improvement for a specific task. The focus here is not to optimise performance, but to identify how the information is preserved through the Neural Pipeline.

Static alpha function synapses are used for the connections as they are the synapses that the behaviour of the architecture has been examined with. In [49]

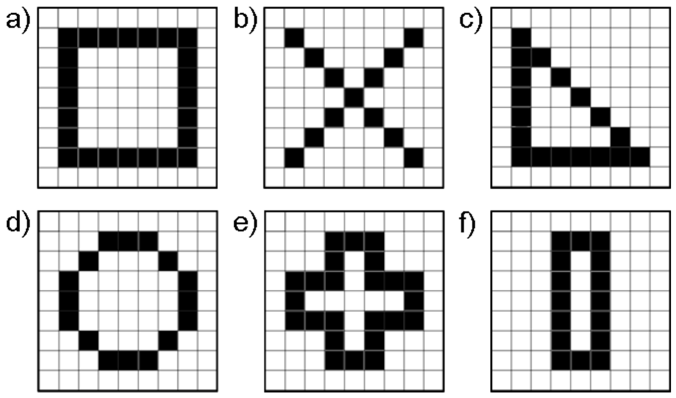


Figure 6.5: The shapes that the pipeline has been trained on (a) square, (b) cross, (c) triangle, (d) circle, (e) plus and (f) rectangle.

they find that dynamic synapses outperform static synapses, for the preservation of information over time. Here the settings for the architecture have been chosen based on tests with static synapses, so static synapses are used despite being the less optimal choice. Information preservation is important in the Neural Pipeline, so replacing these synapses with dynamic synapses and testing the behaviour is a proposal for further work (section 6.6.1).

An overview of the parameters used for the experiments presented in this chapter can be found in table A.7 in the appendix.

6.3 Simple Shape Recognition

While it has been demonstrated that a single LSM can perform pattern recognition [48, 12] and that it is also true for multi-layered LSM [39, 70, 71]; it does not necessarily follow that the Neural Pipeline can. As there are multiple layers and the input is only presented to the first layer the later layers must receive enough of the original data to perform recognition. To demonstrate that pattern recognition is possible using the Neural Pipeline, a simple image recognition task was carried out.

A set of six shapes was produced to fit in the input grid described in section 6.2. These six shapes are shown in figure 6.5. The shapes were chosen as identifiable shapes for a human observer rather than abstract patterns. Hollow shapes have been chosen to reduce the number of input spikes, in turn reducing the level of spiking within the layers and therefore decreasing simulation time. There is intentionally some overlap between the inputs that are black between the different shapes to make the task more challenging for the system. Some have different numbers of black pixels, some the same, to test that the system is not just using the number of active inputs to make a selection.

The objective of this experiment is to test the hypothesis ‘Data presented to the input layer of the pipeline can be recognised at every layer’ using a set of readout neurons to assess the correctness of the response. The limitations of this ability are also considered and extensions to the experiment are suggested.

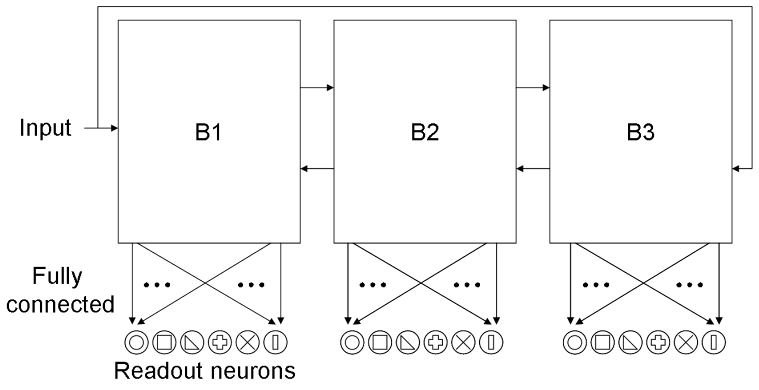


Figure 6.6: The neural pipeline architecture with readout neurons used for learning six shapes.

6.3.1 Method

The Neural Pipeline has been trained to recognise a set of six shapes (figure 6.5) on each of three layers as shown in figure 6.6. The shapes are presented (at 1 spike per ms) for 10ms to layer 1 at the start of the simulation. Before this all neurons are silent. The 81 inputs are connected to 81 of the 100 neurons in the first layer only. There are 100 neurons per layer with 6 readout neurons on each layer, one for each of the input shapes. The simulation is run for 100ms. The training and recognition both take place using perfect copies of the shapes and input train with no noise. The case when noise is added to the input is considered in the experiment in section 6.5.

The readout neurons are fully connected to the layer and the initial weights on these connections are randomised between 0 and the 'internal excitation' value of 0.5. The weights are trained using the delta learning rule (as described in section 2.5.2) to identify the input shapes at a particular time window. The times of all of the spikes that occur in the layer are recorded and divided into time windows of 5ms. The first chronological set of unique windows all with non-zero values is used to train the network. The windows must be unique for each shape so that the system can recognise that pattern as belonging to a single shape. They must be non-zero because with no spikes it is not possible for the readouts to fire.

The readout neurons are trained to spike any number of times when their shape is the presented input, but to remain silent when the input presented is not their shape. So for each shape only one readout neuron will spike. Training is carried out until this is true for all of the readout neurons.

The six shapes have been tested on one set of arbitrarily chosen internal connections. The weights to the readouts are trained based on the activity within the layer.

6.3.1.1 Topology Experiment

To determine that a set of inputs can be learnt with different topologies of internal connections, a smaller experiment was run 100 times. 50 neurons were used per layer, with 10 different inputs of size 40. Of the 40 input bits each pattern had 5 active bits and 35 inactive ones. The reduction in neurons per layer and the number of active input bits (when compared to the shape experiment) was chosen to decrease the simulation time as this experiment was to be repeated 100 times. Only the first layer readout was trained. In each of the 100 runs the connections were varied by seeding the random number generator with different values.

All 100 trials successfully learnt the series of 10 inputs correctly. This is indicative that a specific connection structure is not necessary to allow patterns to be learnt. The average connectivity is important in determining behaviour, as outlined in section 4.6.3.

6.3.1.2 Capacity Experiment

To demonstrate sufficient capacity for the learning experiments in this chapter a preliminary experiment was performed. The experiment was carried out using three layer Neural Pipelines, with layers of size 10 or 20 neurons. The aim of the experiment was to train the system to recognise a number of patterns to identify that the capacity was sufficient for the following tests. This size of network was chosen to reduce simulation time, because the readout neurons are fully connected to the layers so every additional neuron in a layer adds computational expense. The size of input was chosen to be 80% of the neurons in the layer, approximately the same as the size of the grid used for the shape recognition experiments. Five of the input neurons were active at any time. For 10 neurons this gives 56 possible permutations (8 choose 5) and 4368 permutations for 20 neurons (16 choose 5). For 10 neurons all 56 patterns were tested, and for 20 neurons 200 patterns were tested. This was chosen as a stopping point, because it showed a capacity of 200 which is more than sufficient for the following experiments. As the size of the readout layer increases, the simulation time also increases, so it becomes more sensible to have multiple layers with smaller numbers of readouts.

The inputs were only learnt by the first layer, because all of the layers in the system are the same and should therefore have the same capacity. The inputs here could be presented to any of the layers (from any other layer) and would still have the same outcome with regard to capacity. Training all of the layers in this case would only increase the simulation time. The inputs were specifically chosen to have a high degree of overlap, to make telling them apart harder for the system. This means that the capacity is representative of a lower limit, rather than an upper limit.

The results of the experiment are that the 10 neuron system was correctly able to learn all 56 of the possible inputs. The 20 neuron network was tested with up to 200 inputs, and was able to store all 200 of them.

6.3.2 Results

The internal state of the Neural Pipeline when each of the six shapes shown in figure 6.5 are presented is shown in figures 6.7, 6.8 and 6.9. Each unit of the graph represents the number of times that a particular neuron within each layer has fired in the time window.

The active neurons in the first layer are those that receive the input. This means that time window 2 in figures 6.7, 6.8 and 6.9 are equivalent to the original shapes. This can be seen by rearranging the window 2 row in the graph into a 9 by 9 square grid, which will produce the same 6 input patterns as shown in figure 6.5. By the time the shape has reached the second layer it can no longer be seen in its original form, but the system can be trained to associate the new pattern with the correct shape.

The response from the readout neurons is shown in figure 6.10, this shows that the neurons all respond correctly to each shape being presented. The corresponding neuron fires for each shape and all other neurons do not fire. There is no reward or punishment for firing multiple times, so the presence of 2 spikes in layer 3 for circle is no better (or worse) than the single spike for any of the other shapes. The graph demonstrates that all three layers correctly recognise each of the six shapes.

6.3.3 Discussion

The results outlined in section 6.3.2 show that it is possible to train a 3 layered Neural Pipeline to recognise a set of six shapes on each of the three layers. **This is a proof of concept that the information can be passed through the pipeline and can still be classified by the final stage. The secondary section of the experiment illustrates that recognition is possible independent of the internal topology, this is a property of LSM [49]. The advantage of identifying the input patterns on each of the layers is to output the historical sequence of inputs in order.** The layers can preserve the relative timing of the inputs as they are presented. Each layer can respond to the same property as shown here, or to different properties as addressed in the following section.

An inevitable consequence of randomised networks is the degradation of information from the input pattern as the activity propagates through the system. This is shown in [56] for a single network and in section 5.3 for the Neural Pipeline. With this knowledge the following questions are provoked: ‘how many layers can information survive through?’ and ‘how do the parameters influence this?’. In the Neural Pipeline the decrease in separation between distinct inputs may be magnified by each layer, because they are chained and the input is only presented to the first layer. The parameter choices made for the Neural Pipeline are likely to influence the change in separation over time. Identifying this relationship is suggested for future work in section 6.6.1. It is possible that the readouts from the earlier layers may be used to improve the response of later layers, to reduce the impact of the inevitable loss of information.

The capacity of the Neural Pipeline architecture has been demonstrated to be large enough to carry out extensions to the experiments presented here. Further tests are needed to determine the limits of the capacity with different sizes of layer, but it is likely that they will depend on the data to be stored and the learning algorithm used. These extensions are discussed in the further work section 6.6.1.

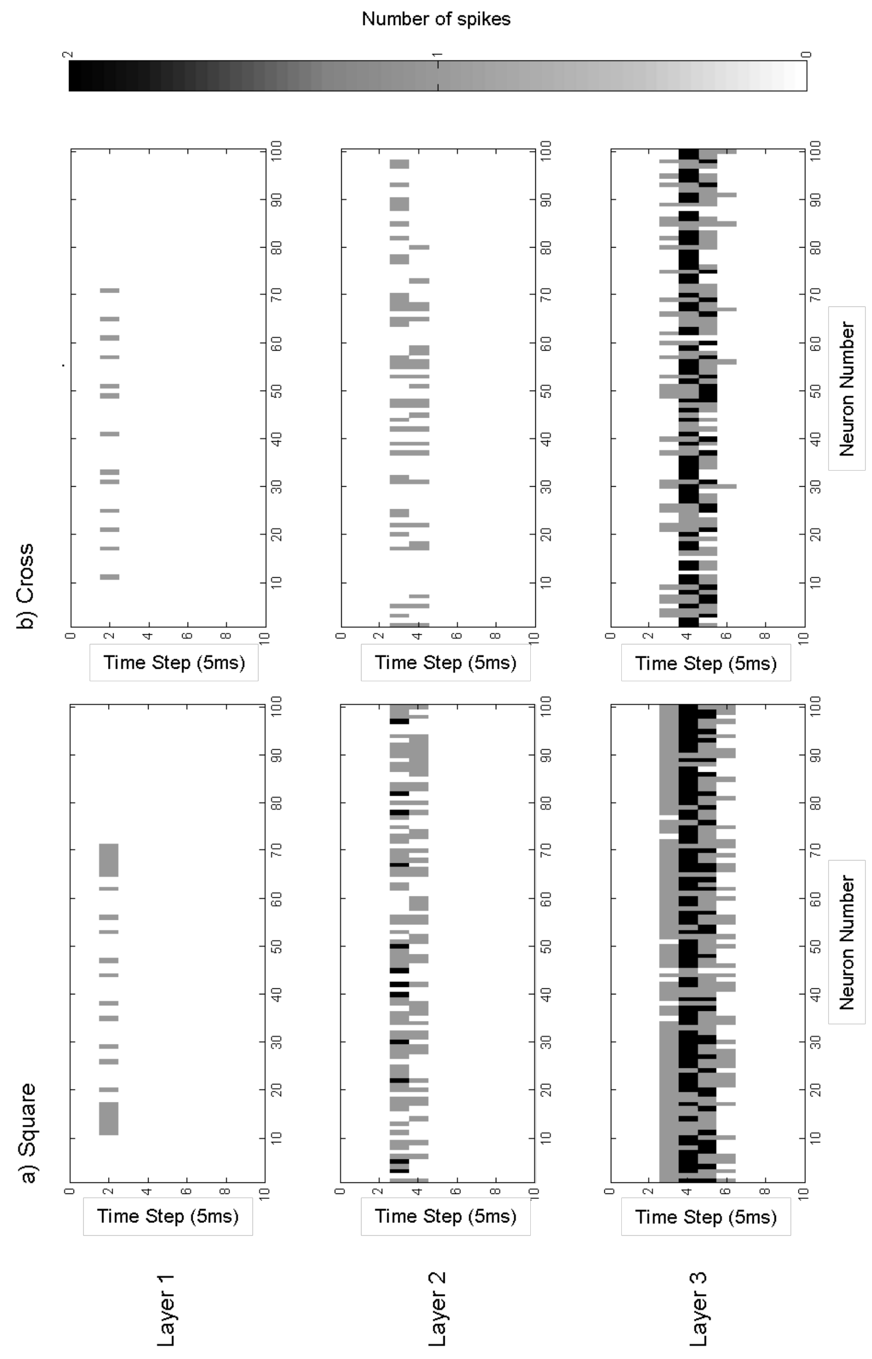


Figure 6.7: The response of each layer in the Neural Pipeline when an input shape is presented. These graphs show the output for Square and Cross.

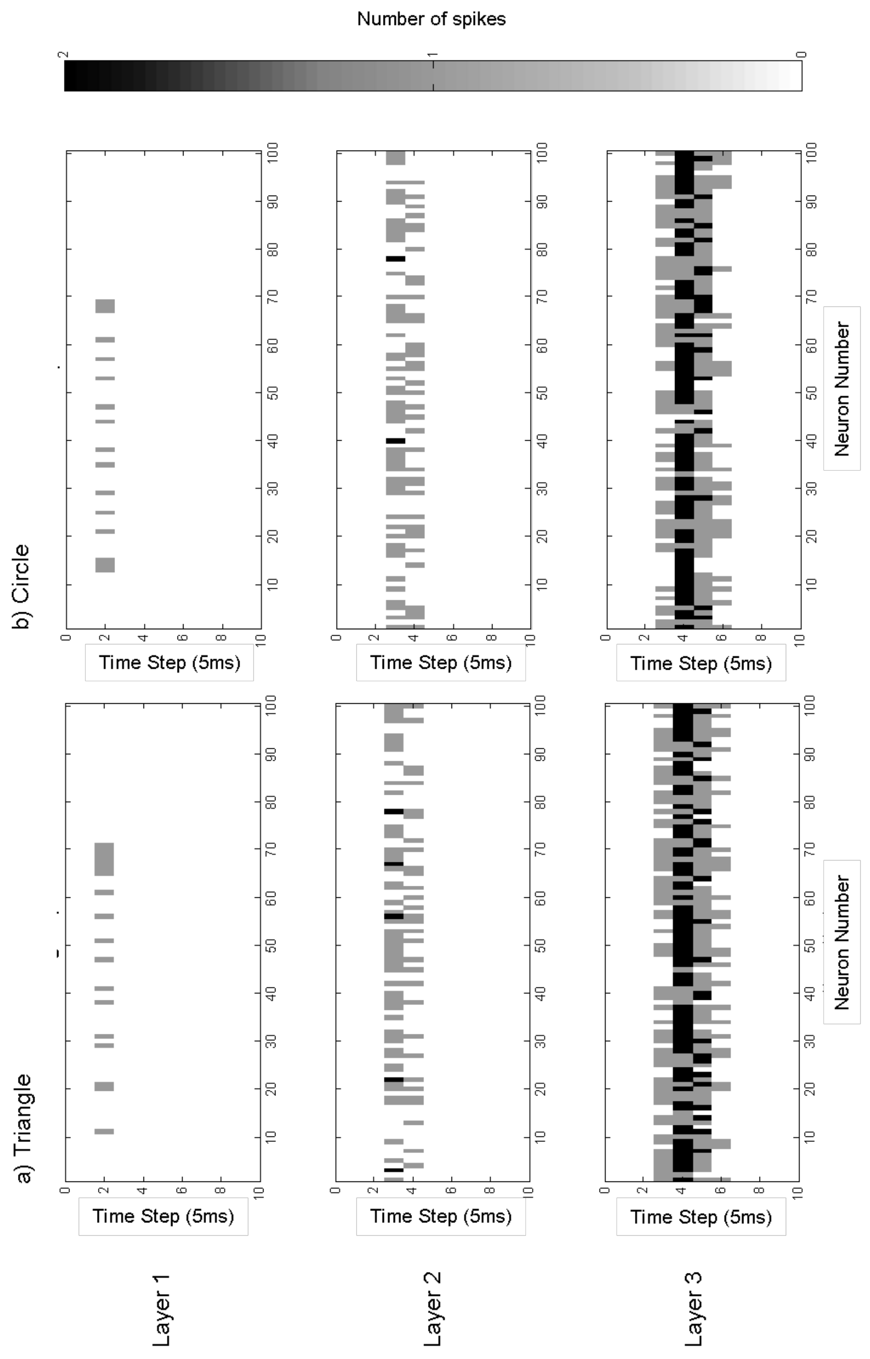


Figure 6.8: The response of each layer in the Neural Pipeline when an input shape is presented. These graphs show the output for Triangle and Circle.

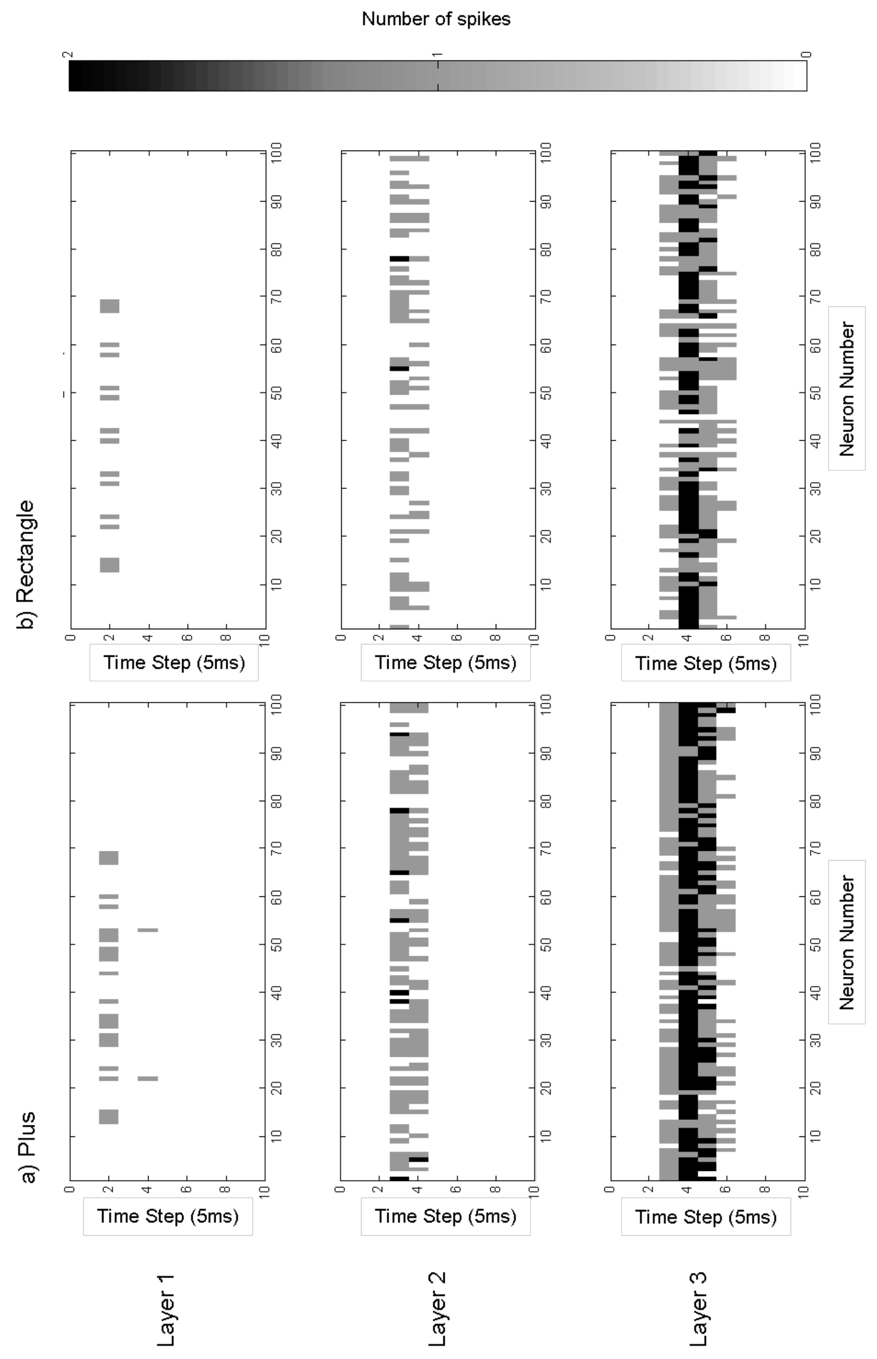


Figure 6.9: The response of each layer in the Neural Pipeline when an input shape is presented. These graphs show the output for Plus and Rectangle.

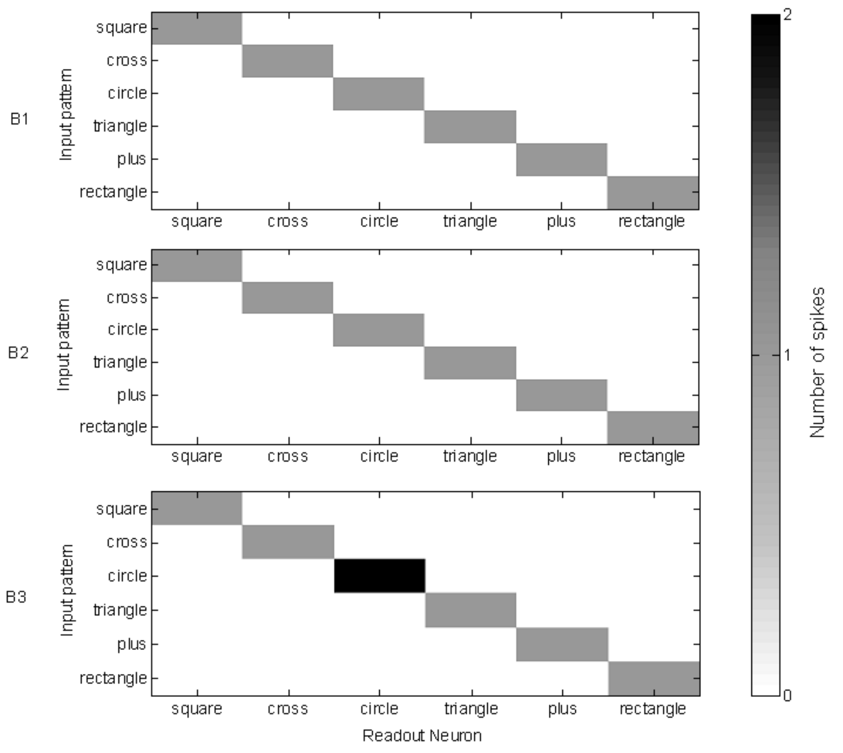


Figure 6.10: The spike totals from the readout neurons from the shape experiment.

6.4 Position

The positional experiment expands upon the shape experiment outlined in section 6.3, but rather than testing whether the same shape can be identified at each layer, this experiment tests the following hypothesis 'Different layers can recognise different properties of the same input'. This is a key aspect in the functionality of the Neural Pipeline as an architecture for use in pattern recognition, because the different layers perform different processing tasks. They will each carry out a distinct task rather than repeating the same one.

The two features being extracted in this experiment are the shape and the position. The shape is chosen because it was the focus of the previous experiment, the position is added as a second feature because it can be achieved easily using the 81 pixel black and white grid. Alternative properties include orientation, size, colour and movement. The reasons for selecting position over the other alternatives are outlined below. The limitations are not general limitations of the architecture but of this particular choice of input grid.

Colour would require a more complex encoding than inputs being on or off (spiking or not spiking) as is the case for the black and white grid. This is discussed in the further work (section 6.6.1). Movement would require a much longer input and thus much longer simulation time, making it an undesirable choice for this experiment. Using a small grid such as this the orientation is not suitable, because the coarse grained pixels would provide very few possible orientations and the shapes

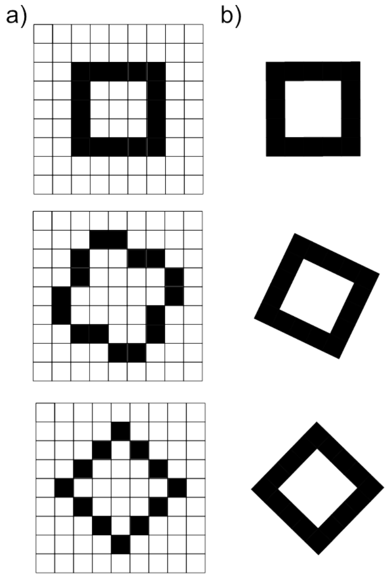


Figure 6.11: a) Three possible rotations of a square, constrained by the 81 pixel grid. b) How these squares would look if not constrained by the grid.

would be deformed by reorienting them. An example of this is shown in figure 6.11, three rotations are shown as constrained by the grid (column a). Only the top image matches the unconstrained version (shown in column b) very well, the lower two shapes appear dissimilar because of the low resolution of the grid. The issue is that with this square grid, two shapes which appear different initially (e.g. square and circle) once rotated can start to look much more similar. This makes rotation a poor choice in this instance.

Resizing the shapes is the most suitable alternative to position, however there are fewer possible sizes than positions with the 81 pixel grid. The granularity of the grid will cause problems as with the rotation, making some of the shapes deform as they are resized. An example of this is shown in figure 6.12. The circle (column b) changes its shape as it is rescaled. Although the length scales linearly, (reducing by 2 pixels each step) the proportions of the sides compared to the curved section changes. The cross (column a) retains the same proportions for each of the three sizes. The variance in how different shapes are influenced is a problem, as well as the deformation itself. The position is independent of the shape, therefore position has been chosen as the preferred option.

The shapes shown in figure 6.5 are too large to reposition on the grid to provide different positions. Therefore two shapes (cross and square) have been shrunk to 3 by 3 pixels to reposition around the grid. Cross and square were chosen because they are still easily recognisable at this size. Five different positions have been used for each shape, they have been chosen with no overlap to make the recognition task simpler.

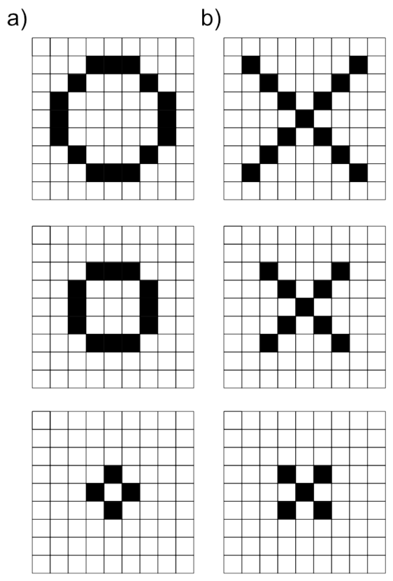


Figure 6.12: Three different sizes of circle (column a) and cross (column b). Cross scales well, each of the three images have the same shape, circle scales poorly each of the three have slightly different shapes.

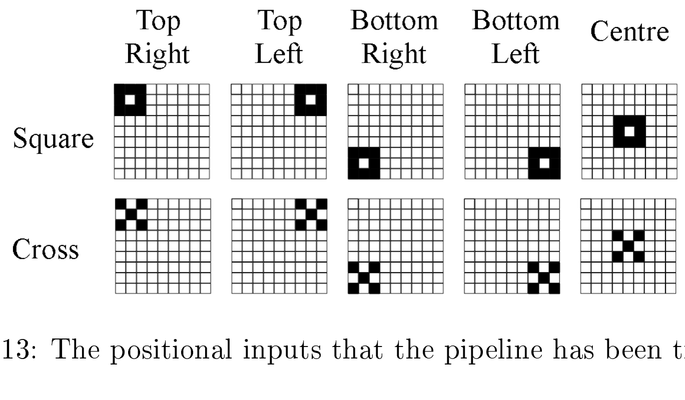


Figure 6.13: The positional inputs that the pipeline has been trained on.

6.4.1 Method

The inputs provided to the system use the same 9 by 9 grid as the inputs in the shape experiment (section 6.3). Here, two of the same shapes (cross and square) are used and are shrunk and repositioned about the input grid, in five specific locations.

The ten different images shown in figure 6.13 are presented in the same way as the shape experiment (section 6.3) as an 81 pixel grid. The inputs are again presented to layer 1 over the first 10ms of the simulation at 1 spike per ms. Each of the 81 pixels from the grid is connected to a different neuron in the first layer. As with the shape experiment there are 100 neurons per layer, but there are now five readouts to recognise the position on layer 1, and two readouts to recognise the shape on layers 2 and 3. The layers were allocated with position first and shape second based anecdotally on the idea of a person (or animal) noticing that 'something' is in a particular position, before identifying what it is.

As the architecture is a computational system rather than an evolved system it is not constrained to have the attributes in this order. As discussed earlier 6.3.3 the information degrades over time, so it may be beneficial to extract more subtle attributes in the earlier layers, but this is not a strict requirement. This means that the system is flexible and can be adapted to the requirements of the user, with the attributes being addressed by their layer of choice.

The readout neurons are fully connected, with weights initialised to between 0 and 0.5 and trained using the delta learning rule. The system is trained using time windows as explained in section 6.2. The readout neurons are trained to spike only when an image of their shape or position is presented, and not to spike for any of the other shapes or positions. The corresponding readouts are allowed to spike one or more times when the correct image is presented. The presented inputs are noiseless and the simulation is run for 100ms. All of these decisions are common between the experiments and justification of the choices is given in section 6.2.

6.4.2 Results

The results of the positional experiment are shown in figure 6.14. From the graph it can be seen that the first layer can correctly identify the position of the input in each of the five locations, independently of shape information. Layers 2 and 3 are able to recognise the shape independent of the position. As described for the shape experiment (section 6.3.2) the presence of multiple spikes from the correct readout neuron (in this case in layer 3) is not better (or worse) than a single spike.

The results show that using inputs without noise, in specified positions, it is possible to identify both the shape and position different layers of a three layered Neural Pipeline.

6.4.3 Discussion

The system could correctly identify all positions and shapes correctly when each of 10 images (5 positions for 2 shapes) were presented. /textbfThey represent a proof

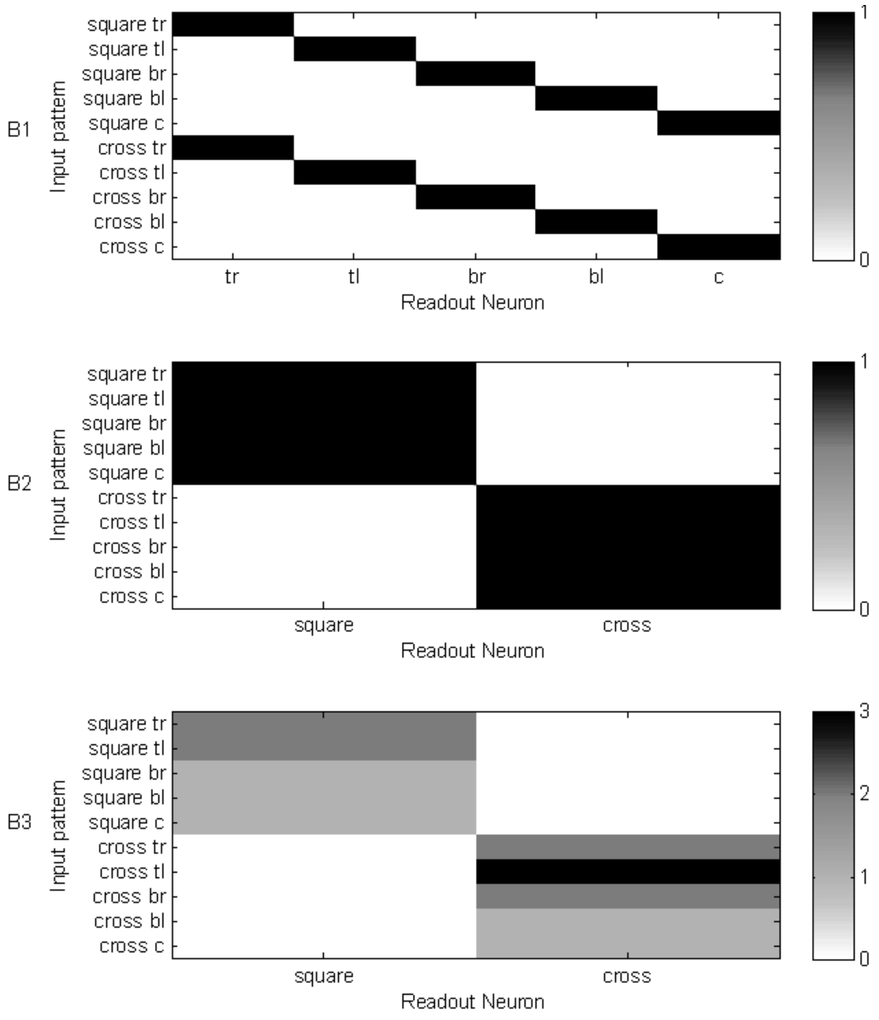


Figure 6.14: Results when a three layer pipeline is trained to recognise position using layer 1 and shape using layers 2 and 3.

of concept that it is possible for different layers to identify different properties of the input pattern.

The results presented here do not show position invariance, because only five specified positions are used. To test this the input shapes could be presented at any location on the input grid. This needs to be addressed in future work, see section 6.6.1 for more details. Other extensions include increasing the number of input properties investigated and testing for how many layers the information is identifiable. This may vary depending on the property (because some may be harder to identify than others, and require more separation). Further tests are required to suggest a reasonable order for the properties, e.g. position on the first layer, shape on the second and colour on the third.

6.5 Introducing input noise

The inputs and training used in the experiments described in sections 6.3 and 6.4 have no noise. They test whether signal information is present throughout a Neural Pipeline of three stages. Noise was not added because the aim of the experiments was not to test robustness. The results of these experiments show that the information is available in all three layers, so this experiment addresses the system when noise is applied. The hypothesis to be tested is 'Noisy system inputs should cause a system trained on perfect inputs to fail gradually, depending on the magnitude of the noise'. The hypothesis was based on the idea that a small amount of noise will produce a pattern quite similar to the intended one, but with larger amounts of noise the pattern will be more dissimilar. This is expected to produce a gradual failure rather than a sudden failure.

In a single layer LSM in [49] noise is added to the signal by shifting each of the input spikes by an amount sampled from a Gaussian distribution, they call this shift 'jitter'. Each of the original spike times are used as the mean value for a set of Gaussian probability density functions (PDFs), a new spike time is then chosen using the Gaussian PDF to construct the noisy signal. In their speech recognition experiment, a pool of 50 readout neurons is used for output and the overall activity of the readouts is used to determine the response. This is a more robust method than a single neuron per input. The aim of this experiment is to investigate whether it is possible to recognise a noisy signal using only the single neuron per shape and strict response (no firing for inputs that do not match) from the shape experiment (section 6.3).

In the shape and position experiments the input is a consistent 10 spikes at 1ms intervals over the first 10ms, provided as input to all input neurons at the same time. Five different methods of applying noise to the input have been investigated and the inputs used for training have been varied. Comparing each of these methods shows whether noise corrupting the synchronisation of inputs or the rate of inputs has the biggest impact. The tests consider completely synchronised input spikes across all neurons with different rates (in spikes per ms), synchronised inputs with the same rate but different timing, unsynchronised inputs with the same rate and two combinations of unsynchronised and synchronised spikes. It is expected that either the rate or the synchronisation will have a larger impact on the degradation of the output, but it is not possible to predict which.

The simplest form of alternative input is to provide a consistent spike train, one every ms, over a different number of ms. This is not a noisy input, but is different to the input used for training. It illustrates how tolerant the system is to the wrong magnitude of input. The second applies the Gaussian noise used in [49]. When the method is applied to the Neural Pipeline input, each of the 10 input spikes is moved to a new position chosen from a Gaussian PDF with a mean value of the original spike location. The original locations are shown in figure 6.15 diagram (a) and the Gaussian distributions that the new spike times are chosen from are shown in figure 6.15 diagram (b). The standard deviations of the Gaussian distributions are varied in experiment 2 'Gaussian shifted spike train' (see the following section). The overlap that the Gaussian distributions have with one another depends on the standard deviation. If there is overlap then the order of the spikes can be changed, as shown in the example spike train in figure 6.15 diagram (c), where spikes 4 and 5 have switched. This makes no difference to the simulation, because it just receives a list of spike times. Negative values of time are not permitted, so if any of the Gaussians extend below 0 they are given a positive value instead. This is shown in Gaussian 1 figure 6.15 diagram (b), and the resulting spike 1 in figure 6.15 diagram (c).

In both of these cases (a different length regular spike train and a Gaussian shifted spike train) all input neurons receive the spike train at the same times. To test whether this synchronisation has an influence, the next stage of the experiment uses different spike trains for different input neurons. For ease of implementation a Poisson spike generator was used. This produces a spike train with an average number of spikes per ms, chosen from a Poisson distribution. The resulting train is similar to the Gaussian shifted spike train. This distribution is chosen to reduce the run time of the simulation.

The final part of the experiment uses a combination of a Poisson and regular spike train. This tests the extent to which the system, using this readout structure and learning method, requires the input neurons to receive their input spikes at the same time. Two types of combination are used. The first combines a 5ms regular spike train with different Poisson trains to all neurons. It means that every neuron receives a number of spikes synchronised with the other neurons and a number of spikes that are unsynchronised. The second combination applies the 10ms regular spike train to a proportion of the input neurons and different Poisson spike trains to each of the other input neurons. Testing these two different combinations allows a comparison between the case where all neurons receive a slightly noise corrupted signal (combination 1) with one where some neurons receive a perfect input and others a noisy input (combination 2).

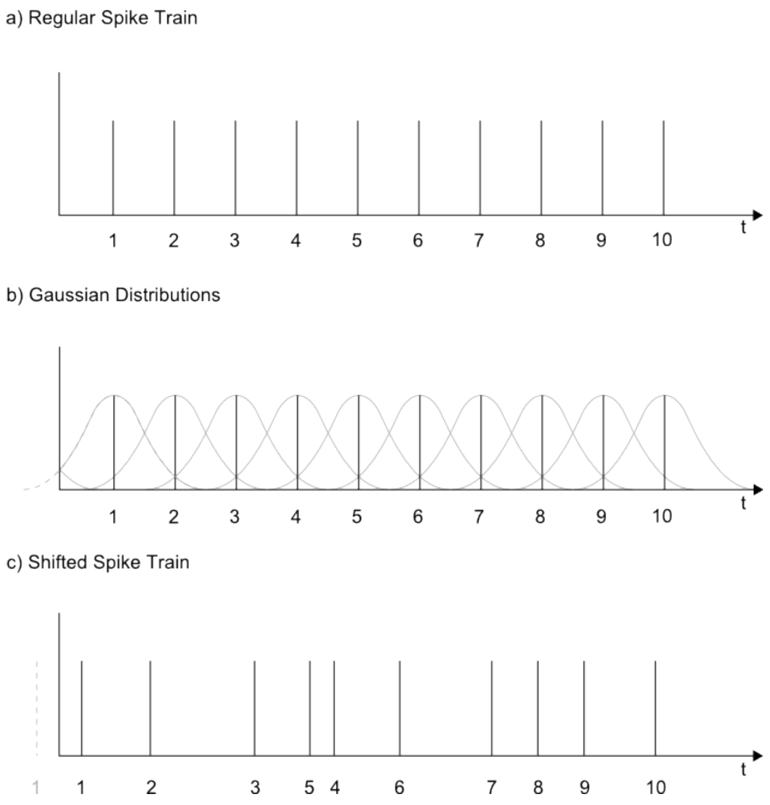


Figure 6.15: (a) shows the original input with spikes every ms, (b) is the same regular train with Gaussian distributions overlaid. Negative times are not allowed, so Gaussians which go below 0 are flipped about the y axis, this is illustrated on the first Gaussian in (b). From these distributions a new spike train is chosen, an example of this is shown in (c). The overlap means that spikes can swap positions, such as 4 and 5 in (c), this is not a problem because the input is just a list of spike times.

6.5.1 Method

The experiment looks at how different types of input influence the response of the system. The five different types of input are summarised below:

1. **Regular spike train** - a train of 1 spike per ms is presented to all neurons. This is 'noisy' because the tests use different durations to the one used for training.
2. **Gaussian shifted spike train** - 10 spikes are shifted about Gaussian PDFs with mean values corresponding to the original spike train.
3. **Poisson spike train** - Different Poisson spike trains are presented to each of the input neurons.
4. **Poisson and Regular combination** - A regular spike train and a Poisson spike train are combined to provide input to all of the neurons.
5. **Poisson and Regular split between inputs** - A regular spike train is presented to some of the input neurons and a Poisson spike train to the others.

For all of the tests the system is trained on a 10ms regular spike train. In additional a variant trained on a Poisson spike train is provided for the Poisson tests. The Poisson spike train used in training is different to the one used in the simulation run. The parameter values and settings for each of the different inputs are outlined in the following sections.

A single run is used to test each of the regular spike trains. This is because the same internal connections are used and the regular spike trains have no variation. The result for one run is therefore the same as a number of runs averaged.

The experiment using a Gaussian shifted spike train is repeated 100 times for each of 7 different standard deviations of distribution. A number of runs are carried out because each run is different, so the average behaviour is considered. 100 runs was chosen to generate enough runs to produce a range of inputs while keeping simulation time reasonable.

The Poisson tests represent examples of individual runs. These illustrate the type of impact noise has on the input The decision was taken not to run many repetitions of these experiments to minimise the time taken, so that different combinations of Poisson trains and regular trains could be run instead. From these runs, suggestions for more complete experiments using noise have been suggested (section 6.6.1).

All experiments are conducted using the same Neural Pipeline architecture as used in the shape experiment (section 6.3), so that the results can be compared.

Two measures of correctness are defined to compare the influence of the noise; a correctness value and a 'perfect' count. The 'perfect' count is the simplest measure, it is the number of input shapes that produce an output with only their corresponding neuron firing, as this is considered a perfect response. This produces an integer value between 0 and the number of inputs (6 in this case). The correctness value is a score of +1 when the highest spiking readout is correct, 0 when there is no response from any readout and -1 when the highest firing readout is not correct (or there is more than one with the highest firing rate). This produces a score of between plus and minus the number of inputs (here plus and minus 6). The 0 case is not counted as positive or negative because it is considered better for the system to respond that it cannot identify the shape, rather than responding incorrectly. The case where multiple readout neurons have the highest firing rate is given a negative score because even if one of the highest firing readouts is the correct one it is not possible to tell this from the system output. These scores are calculated for all of the layers, because each layer has its own independent set of readout neurons.

The results of each of the experiments are presented and described in the following sections. In all graphs the shapes are abbreviated to their numbers: 1 Square, 2 Cross, 3 Triangle, 4 Circle, 5 Plus and 6 Rectangle, to save space. A discussion of these findings is provided in section 6.5.

6.5.2 Regular spike train

For this experiment a regular input was provided, with 1 spike every ms. The difference from the input in the shape experiment (section 6.3) was the number of ms that the input was provided for. The different durations used for the test are all of the trains between 5 and 14ms. 5ms was the lowest number used because it produced 0 spikes in all layers, this means that any smaller input would also produce 0 spikes and therefore no output from the readout. 14ms was the highest input tested because the response was already very noisy at 14ms, so continuing to add more input spikes would not test the ‘gradual failure’ described in the hypothesis.

The system uses the same parameters as the shape experiment and is trained on a 10ms regular spike train.

The correctness values and number of perfect responses for all tested lengths are shown in figures 6.17 and 6.18 respectively. When 5 regularly spaced spikes are presented as input there is no activity at any stage of the architecture. All neurons (and therefore readouts) remain silent. This means that the correctness value and perfect runs values are 0, and it sets a lower limit on how many spikes the system must receive in order to produce an output. This limit is controlled by the system parameters, such as the internal excitation.

The results for durations of 6,7,8,9 and 10 all behaved completely correctly, this is shown by the perfect and correctness values of 6 in all layers. Their responses are identical to the graph shown in figure 6.10.

The results from 11 to 14 ms are shown in figure 6.16. These graphs and figures 6.17 and 6.18 show that the correctness and number of perfect runs reduces as the number of input spikes is increased. For all of these values the correctness value is better in layer 2 than in 1 or 3. This suggests that there is a trade off between too many spikes causing many of the readouts to become active and information loss through the system. Layer 1 has more readouts spiking than layers 2 or 3, this can be seen in figure 6.16, this is because the layer is being provided with more spikes than it was trained with, so the readouts are too sensitive. It suggests that the spikes are becoming more spread out by the time they reach the second layer, causing fewer

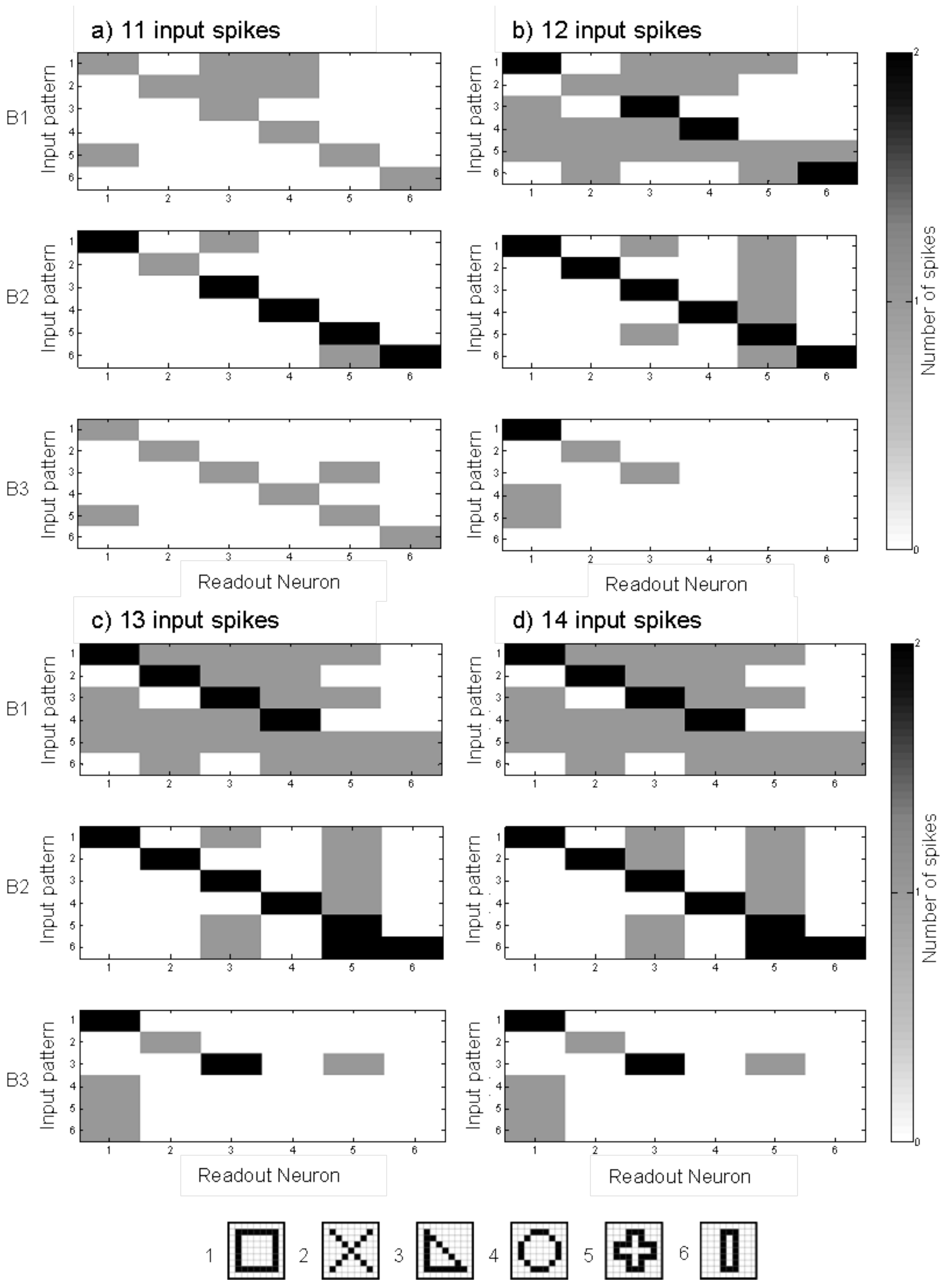


Figure 6.16: The response of the readout neurons when applying more regularly spaced input spikes than in training, from (a) 11 to (d) 14.

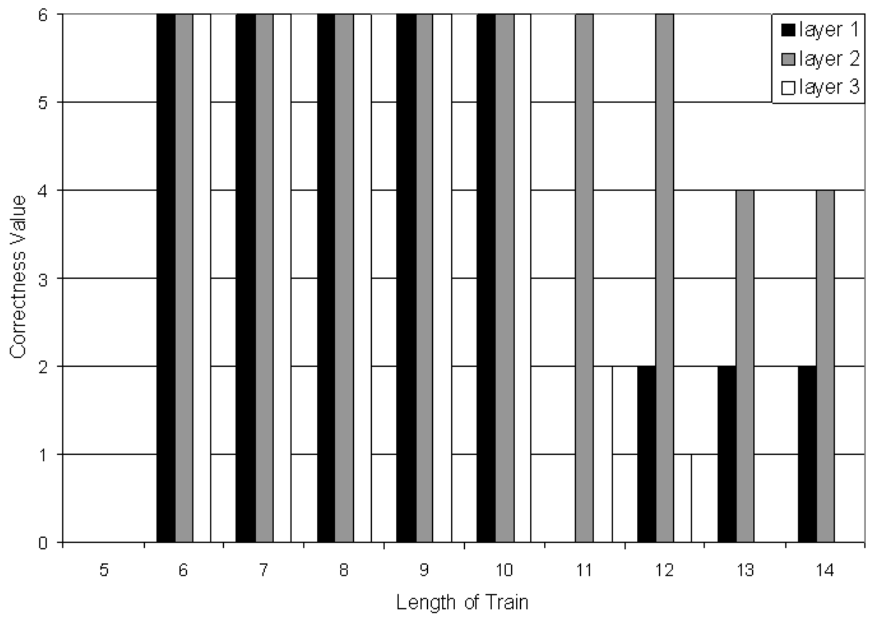


Figure 6.17: The correctness value for each of the runs using different lengths of regular input, of between 5 and 14 spikes. Correctness is defined in section 6.5.1.

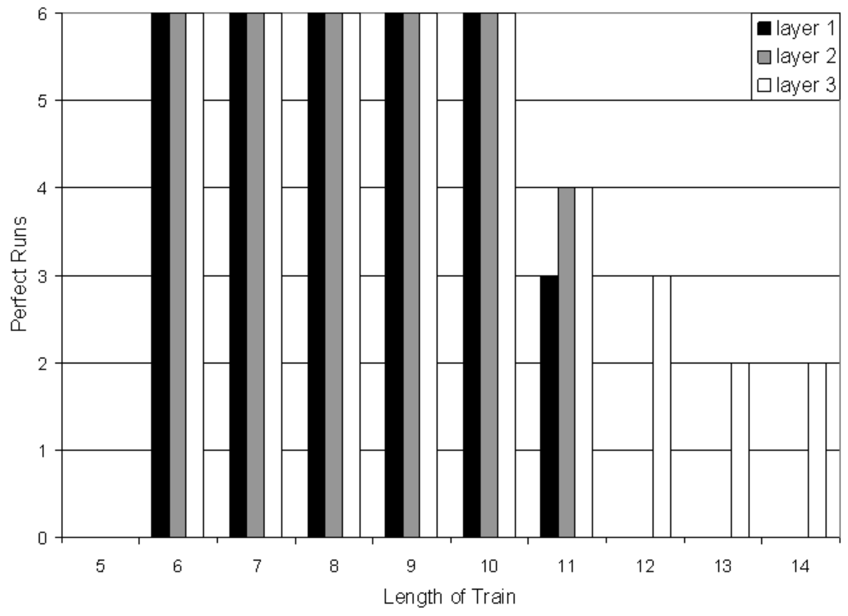


Figure 6.18: The number of perfect responses for each of the different lengths of regular input, of between 5 and 14 spikes.

of the readout neurons to spike. To test this theory properly an extension of the experiment using more layers (described in the further work section 6.6.1) would need to be run, to make sure that the peak is at a mid point in the system, rather than there being multiple peaks or an improvement in later layers. It is not possible to rule this out using only three layers.

When only considering ‘perfect runs’ (figure 6.18) layer 3 performs best when the number of input spikes is greater than 10. This is because by layer 3 the readouts spike less than for the other layers and a ‘perfect’ response does not spike for any shape but its own.

6.5.3 Gaussian shifted spike train

Each spike time is chosen using a Gaussian distribution with a mean value of the original spike time from the regular spike train. A Gaussian PDF is used to choose new spike times for each of the 10 original spikes. This process is described further in the introduction section 6.5. Standard deviations of 0.25, 0.5, 1, 2, 4, 8 and 16ms were used for the Gaussian distributions. This provides a range of values where those with the lowest standard deviation produce spikes that are likely to be close to their original locations, but with the higher values they are more likely to be further away. 0.25 was used as the lowest standard deviation because all 100 runs were perfect for layers 1 and 2 and 85 were perfect in layer 3 (with the other 15 having scores of 5, meaning only one shape without a perfect response). This was considered to be a close enough response to the noiseless input to be used as a minimum value. 16 was chosen as the final standard deviation because it is the first value that has complete failures in the first layer, with some of the runs not correctly identifying any of the shapes. This represents the end of the ‘gradual failure’ from the hypothesis.

A shifted version of the original 10ms spike train was produced using the method described in section 6.5.3. This was run 100 times with 7 different values of standard deviation. The perfect runs and correctness values are shown in a series of box plots in figure 6.19. All of the perfect responses are shown in (a) and the correctness values in (b), with one graph representing 100 runs using the standard deviation stated. The mid-line of the box represents the median value, and the top and bottom the quartiles. Whiskers are 1.5 times the interquartile range, with outliers plotted separately.

Examples of some of the incorrect runs are shown in figure 6.20 to illustrate the type of mistakes that the system is making.

The hypothesis to be investigated for this series of experiments is that the system will fail gradually as more noise is applied. The results from this Gaussian shifted experiment demonstrate that for this type of noise the hypothesis is true. In figure 6.19 it can be seen that as the standard deviations of the Gaussians are increased, the number of perfect runs and the correctness value tend to reduce. This can be seen most clearly in layer 3, but also by the outliers (marked with a plus) in layer 2.

Other patterns, too, emerge from the data. The performance decreases in the later layers, with layer 3 having a smaller median value for both perfect runs and correctness value. This suggests that once noise is introduced into the system at the

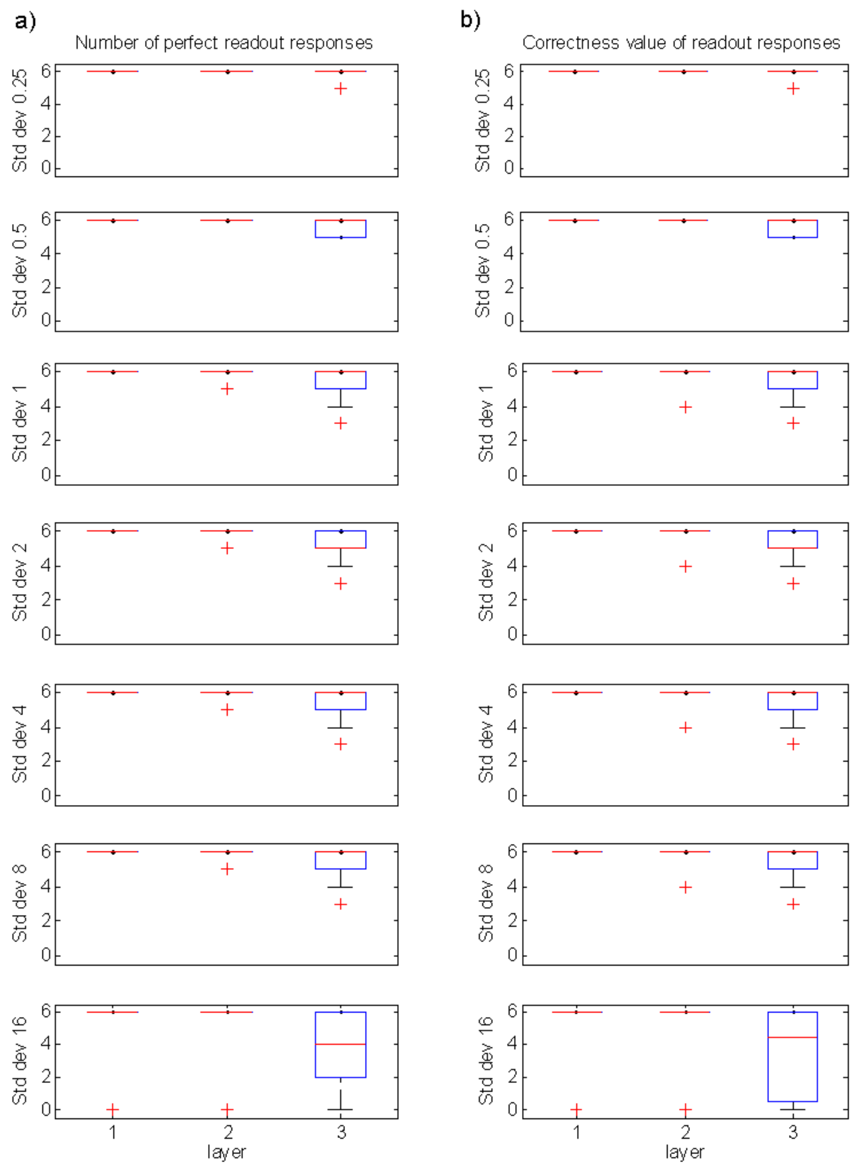


Figure 6.19: Results from running the Gaussian shifted spike train input 100 times for each of the standard deviations. Column (a) shows how many responses are perfect, (b) the correctness value.

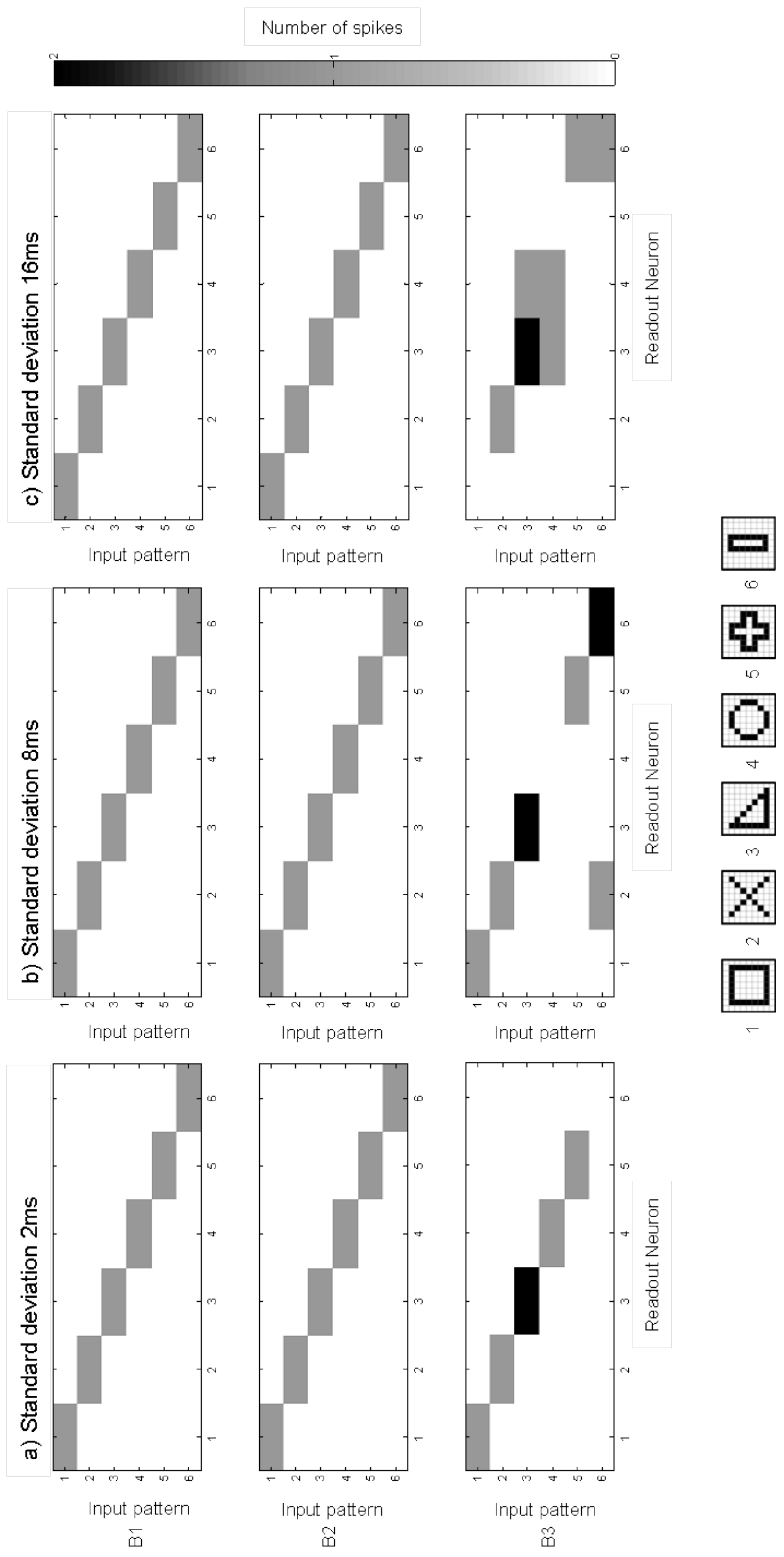


Figure 6.20: Examples of incorrect responses from Gaussian shifted spike trains with different values of standard deviation.

input, the impact is increased the longer the signal is in the system. It could be that the noisy signal diverges more quickly from the original signal, making it more difficult to identify.

The system is still reliable on the first layer until the standard deviation reaches 16, because all 100 runs are perfect for standard deviations of between 0.25 and 8. For layers 1 and 2 the majority of responses are still perfect even with a standard deviation of 16. Even on layer 3 the median number of perfect responses is 4 out of 6. **This shows robustness in the system. A standard deviation of 8 or 16ms is very large considering the intervals are usually every 1ms, and the spike train itself only lasts for 10ms. This shows that as long as the inputs are synchronised the input spike train can be very different to the training and still produce the correct response.**

Another property that can be seen from the results is that the response on the first layer is either 0 or 6. So for any particular noisy input the system either responds perfectly or not at all. This particular result does not match the hypothesis of gradual failure. In this case (on the first layer) it could be preferable to the system failing slowly, because with a slow failure it is not possible to tell which of the shapes the system is correctly identifying. If the system just stops responding if it does not recognise the shape then this is a good signal to the observer that there is too much noise on the input for the system to perform properly.

6.5.4 Poisson spike train

For this test, each input neuron received a different Poisson spike train over the first 10ms of the simulation, with a mean firing rate of either 1 spike/ms or 1.4 spikes/ms. These rates were chosen because they represent an average of 10 spikes in 10ms and 14 spikes in 10ms, the same as the original regular spike train and the highest regular spike train that was tested in section 6.5.2. This allows a comparison between the two.

Test examples were generated for a system trained on a regular spike train of 10ms as with the original shape experiment (section 6.3). Additionally, some runs were generated with a system trained on a different Poisson train for comparison. The results from both were expected to show similar levels of noise in the readout response, because a regular train is a specific instance of a Poisson train.

The results from two example simulation runs of both spike rates are shown in figures 6.21 and 6.22. For both figures (a) is 1 spike per ms (b) 1.4 spikes per ms. Figure 6.21 is the response when the system is trained on the regular 10ms spike train and figure 6.22 is when the system is trained on Poisson spike train different to those received by the neurons.

When a different Poisson spike train is presented on each of the inputs the response of the system is poorer than when the same Gaussian shifted input is applied to all of the neurons (in the previous experiment). The examples in figure 6.21 show perfect scores of only 3 or 4 on each of the layers, rather than the median value of 6 for all of the layers up to a standard deviation of 8 in figure 6.19. These are only example runs, so it is not possible to provide a complete comparison, but with the

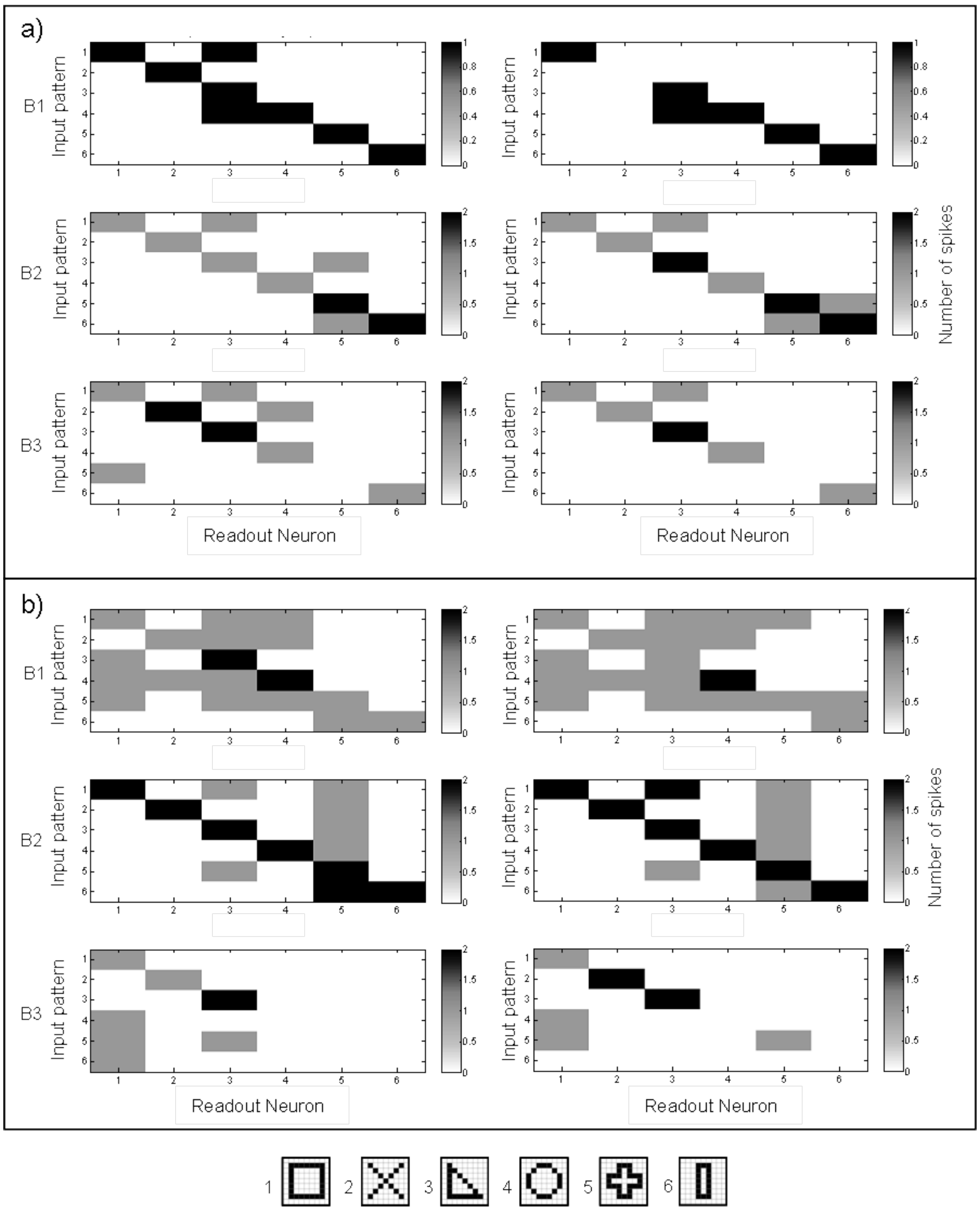


Figure 6.21: Example responses from runs with a different Poisson spike train for each active input, when trained on a regular input. (a) has an average of 1 spike per ms (b) an average of 1.4 spikes per ms.

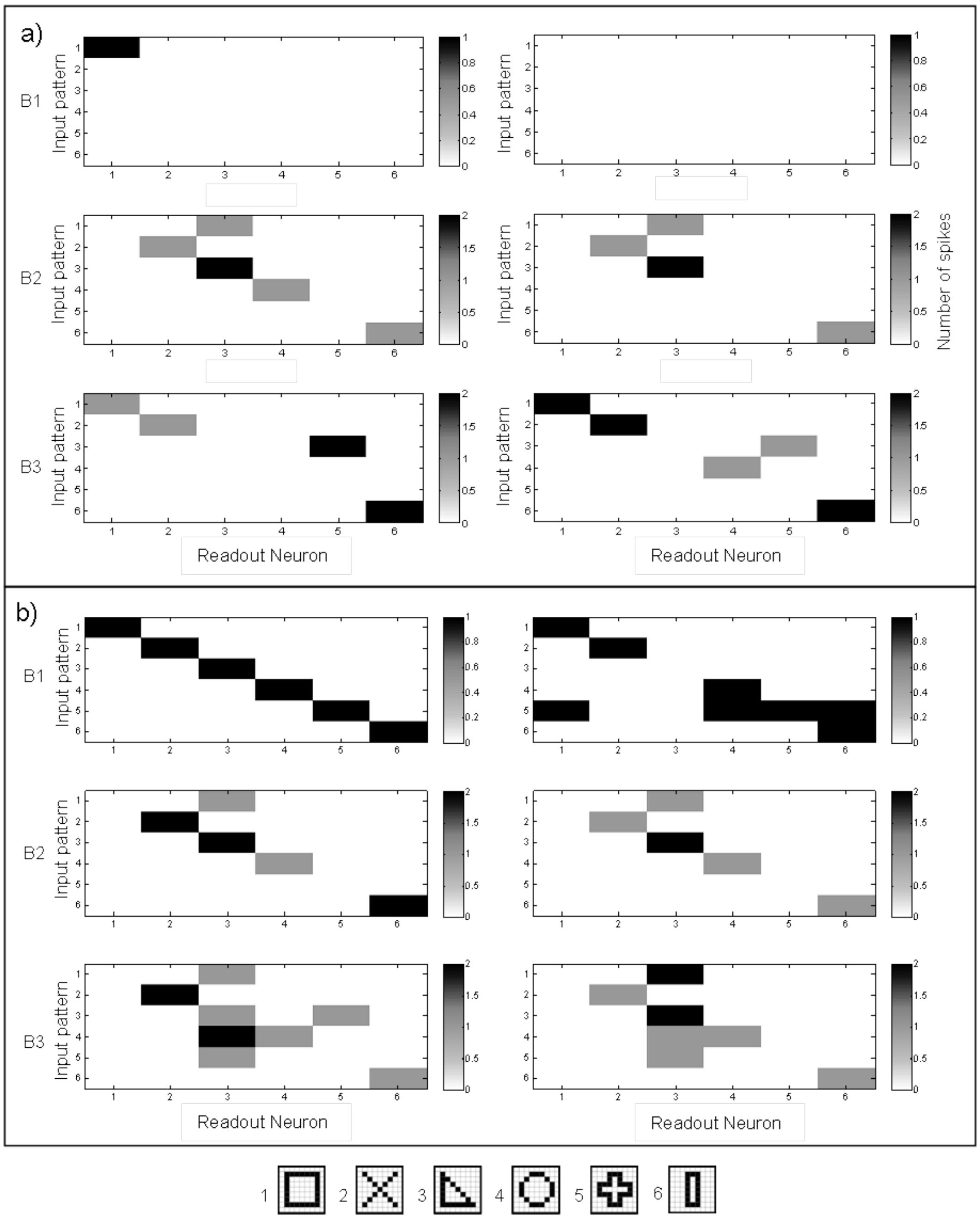


Figure 6.22: Example responses from runs with a different Poisson spike train for each active input, when trained on a different Poisson input. (a) has an average of 1 spike per ms (b) an average of 1.4 spikes per ms.

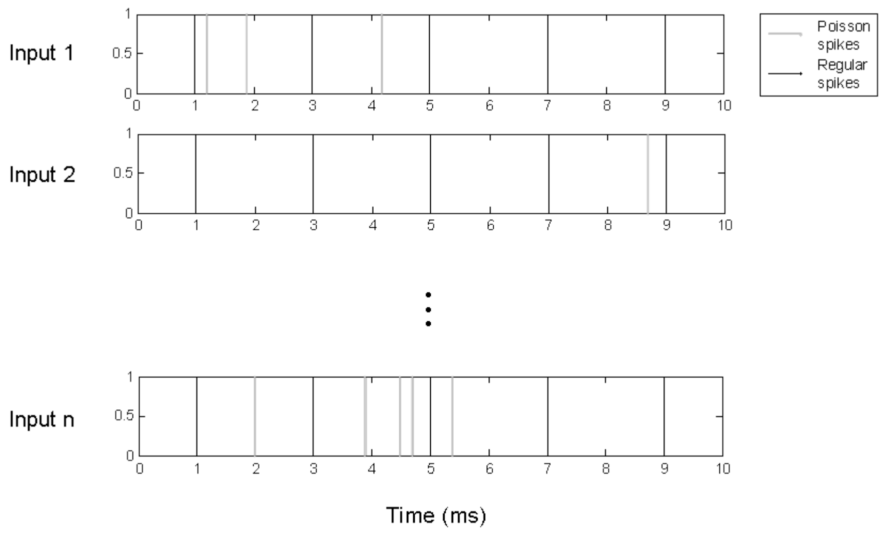


Figure 6.23: An example of the type of spike train used as input when a Regular train and a Poisson train are combined.

Gaussian noise all 100 of the runs had scores of 6 for layer 1 and here both of the examples have a score of 4. **This difference between responses suggests that the synchronisation of the inputs, with respect to each other, is important for the system to correctly recognise the inputs.**

The increase in readout activity seen with 14 regular spikes rather than 10 (figure 6.16) is also seen with a noisy input in figure 6.19. There is also a drop in correctness and perfect responses as the spike rate is increased, as was seen with the regular spikes.

The results using Poisson training were expected to be very similar to the regular training results, but there are some differences. When trained on a Poisson distribution there are fewer spikes on the readout neurons, this may be because there were more spikes in the Poisson spike train used for training, than for testing. To identify whether this is the case the tests would need to be repeated in the same way as the Gaussian experiment.

6.5.5 Poisson and Regular combination

The two combinations of Poisson and regular spike trains are provided to try to understand how the system degrades. The first is described here and the second in the following section. As the results from the previous experiments (sections 6.5.2 and 6.5.3) suggest that the neurons receiving spikes at the same time is more important than the timing itself, these experiments aim to identify a crossover point between them.

Here, each input neuron received five regular spikes at a rate of one spike every

2ms, starting at time 1ms (times 1,3,5,7 and 9ms). In addition to this, a different Poisson train was generated for each of the input neurons. These two trains were combined to provide the input. Examples of these combinations can be seen in figure 6.23.

In this case the Poisson train was reduced to an average spike rate of only 2 spikes over the 10ms input. This rate of Poisson spiking was chosen because the aim of this experiment is to test whether the spikes received out of time cause the readout response to degrade. In the regular spike train experiment (section 6.5.2) it was found that extra spikes caused the system to produce a poor readout response. To separate the presence of 'too many' spikes from spikes that are not synchronised, the Poisson rate is set to be lower than half of the original, regular, spike train (10 spikes). 2 was chosen as a benchmark, so that if the readout response did not suffer much degradation then the number of Poisson spikes could be increased, while remaining below 5.

This input was also tested using a system trained on the original regular spike train and on a Poisson train. This allows comparison with the same results for the Poisson input experiment described above in section 6.5.4.

Figure 6.24 shows two examples of presenting this type of input to a system (a) trained on a Poisson spike train and (b) trained on a regular spike train. When comparing the example results to the regular spike train (section 6.5.2) it can be seen that the system performs less well. The noise causes some of the readout neurons to respond incorrectly. **This suggests that even a small amount of unsynchronised noise, applied to the system input, can cause an imperfect response.**

When compared with the Poisson input graphs in figure 6.22 the combination graphs in figure 6.24 have fewer readout spikes. This may be because the inputs are likely to have fewer spikes in this case, an average of 7 (5 regular and 2 Poisson) rather than 10 or 14 in the Poisson input case.

The two examples have comparable (slightly worse) correctness values to the case when different Poisson distributions are used on all inputs. **This suggests that a compromise of some regular spikes with the Poisson noise is not sufficient to provide completely correct readout responses.**

6.5.6 Poisson and Regular split between inputs

This experiment also aims to identify how important the arrival of spikes at the same time is, to prevent degradation of the readout response. To compare with the combination experiment outlined above (section 6.5.5) an alternative combination method was tested. In this case some of the input neurons receive a 10ms regular signal and others a different 'noisy' Poisson signal.

As the different shapes have different numbers of active input neurons it is not fair to specify a fixed number of inputs that receive either type of signal. Rather a proportion of the input neurons are given each signal. This proportion ranges between 1/2 and 1/12 of the input neurons receiving a different Poisson spike train. Where the numbers do not divide exactly they are rounded down to the nearest

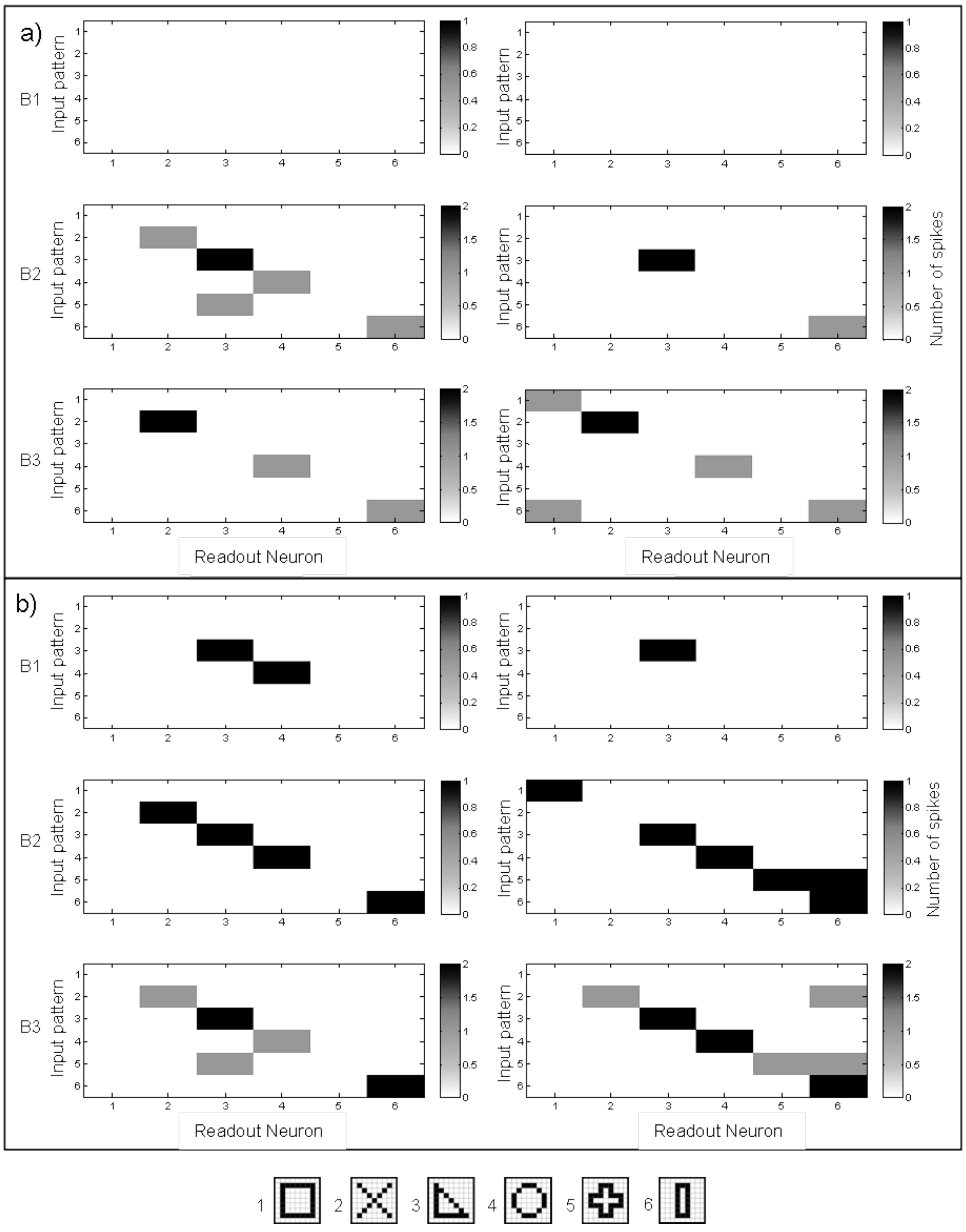


Figure 6.24: Examples of the response when a combination of Poisson spike trains and a regular spike train are applied to the system. (a) shows two examples using a system trained on a different Poisson spike train and (b) shows two examples using a system trained on a regular spike train.

integer.

To identify whether it is the regular spike train or the synchronisation of the inputs that causes the correct behaviour in this case, the same experiment was run using the same Poisson train rather than a regular spike train. Different Poisson trains were used on the other inputs and training was still performed using the regular train.

The higher spike rate of 1.4 spikes/ms was used for both experiments. As only a proportion of the inputs were noisy, it was decided that they should use the higher firing rate.

The graphs shown in figure 6.25 represent example cases when n/2, n/8 and n/12 of the active inputs receive a Poisson train and the rest receive a regular 10ms spike train. Example cases for n/2, n/8 and n/12 of the inputs receiving a different Poisson train while the rest receive the same Poisson spike train are shown in figure 6.26.

The examples shown in figure 6.25 show that **this type of noise follows the hypothesis that increasing the noise will gradually cause the system to fail.** With more of the input neurons receiving different inputs there are more mistakes in the readout. When the number is reduced so that it is only n/12 of the inputs that receive a different input signal, the majority of the responses are correct, though there are still some mistakes.

The response when the same Poisson signal is applied, rather than the same regular spike train, is shown in figure 6.26. This graph shows the same pattern, with lower numbers receiving different inputs having more correct responses. It also illustrates that it is not the precise regular timing of the spikes that causes correct readouts, but the synchronisation of the inputs. It shows this because the responses are comparable in correctness and perfect runs to the version with regular spikes (figure 6.25).

6.6 Discussion

The three sets of experiments have shown that it is possible to train a Neural Pipeline to learn simple patterns. These input patterns can all be recognised by three layers independently. The separate layers are used to produce outputs in sequence of the order in which they were presented. The later layers displaying the earlier inputs, while the new inputs are displayed on the first layers. The architecture has been shown to achieve this performance using different connection topologies.

Information is lost the longer the input remains in the system, the amount of information lost within three layers is sufficiently small that each input pattern can be recognised by all layers. When larger numbers of layers are used input to the later layers could be supplemented by information from the readout neurons from the earlier layers, this may extend the time that the information remains within the system.

The experiments show that different properties can be recognised using the different layers. This is an important property of the architecture. It means that the relevant information about the input can be extracted in a specified order. So in this

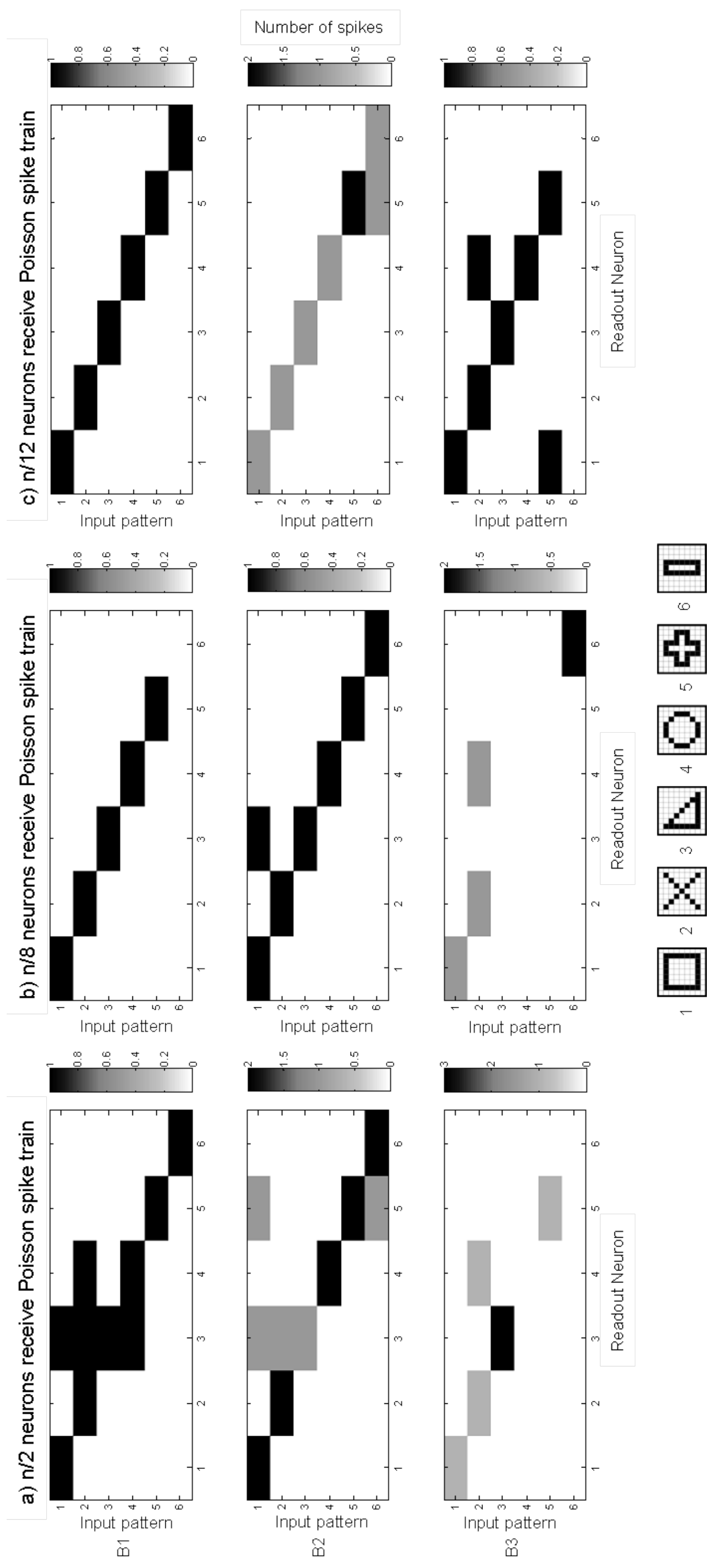


Figure 6.25: The response of the system when different numbers of the inputs receive different Poisson inputs and the rest receive the same regular spike train.

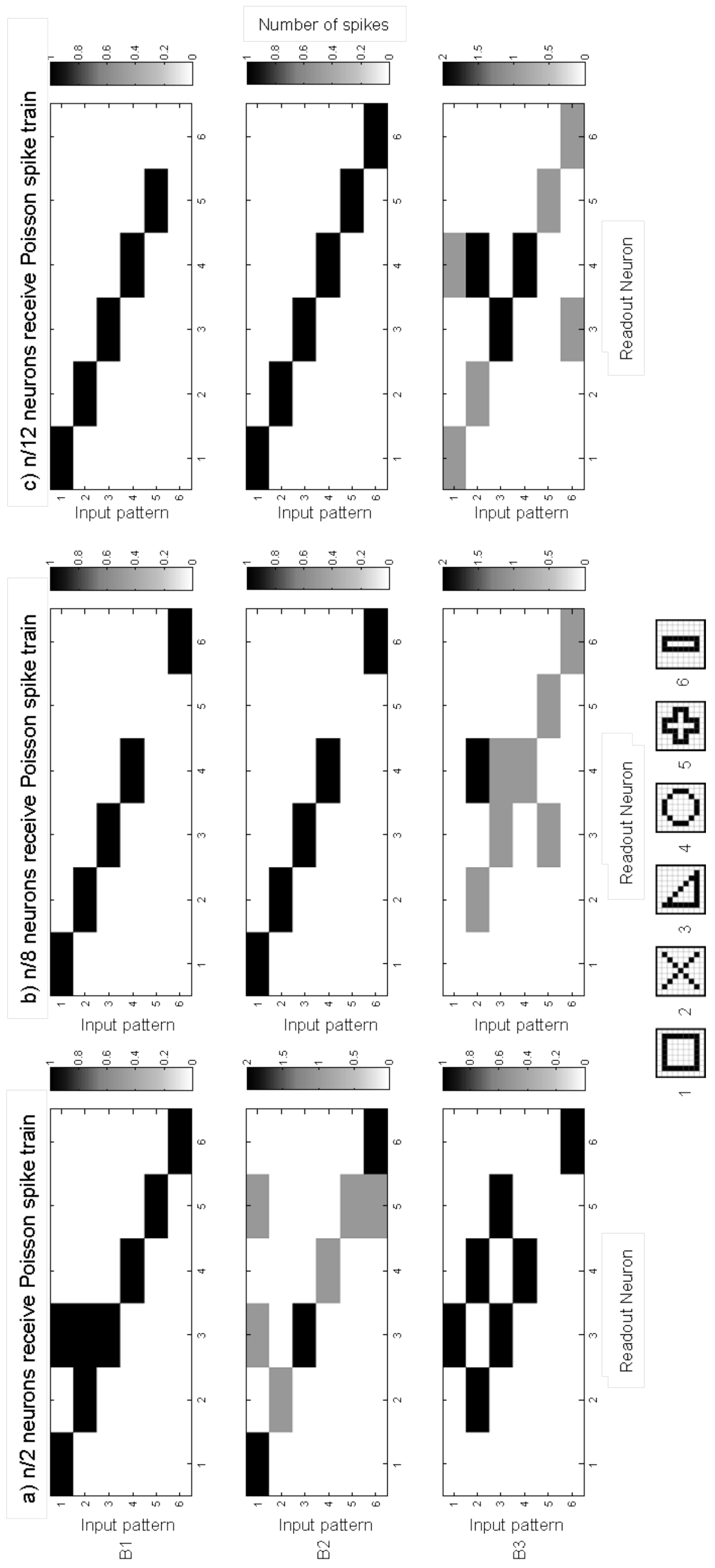


Figure 6.26: The response of the system when different numbers of the inputs receive different Poisson inputs and the rest receive the same Poisson spike train.

example we can identify that ‘something’ is in a particular position, before determining what it is. If the task required the input be identified before its position was established then the layers could be trained to perform this task instead. This ability of the system to handle data in different orders does not match what we would see in biology. The brain has evolved to handle data in a particular order, over millions of years. This system is being trained without the influence of this evolution. The ability is, however, computationally advantageous and as it is a computational system this is a suitable outcome.

This splitting of information between layers may have other advantages the inputs to be learnt could be split between the layers in such a way as to filter out content. The early layers providing a general filter and the later layers being more specialised. Reasons for choosing this method over a single layer are for timing, so that particular inputs are given as output earlier than others and an efficient use of readout neurons. As the readouts are fully connected to the layer, having more readout neurons per layer greatly increases the number of connections and thus increases simulation time. There are multiple ways of splitting the inputs. A coarse grained filter could be applied using the first layer, say to classify into rounded or square shapes. Then a more specialised filter on the second layer, using the readouts on the first layer to inhibit the options that are not possible based on the first classification. One issue with this is that the information content degrades over time as it passes through the layers. There are alternatives that can be tested. One possible solution to the degradation of information would be to enhance the differences between the inputs by training the layers. If this enhancement were present in every layer, then it may be possible to have the data become more focused over time rather than losing information. Another possibility is to filter out the most similar inputs with a low separation value in the first layer, where there is the most information content. Then use the later layers to recognise the more different shapes.

It is possible the different layers, in order to specialise for a particular task, could use different parameters to one another. A good example would be the number of neurons in the layer. To reduce computational load, a layer with a simpler task to perform may be able to have fewer neurons in it than a layer performing a complex task.

It could be argued that based on the results in [49] it would be possible to achieve the same splitting out properties of inputs using a single liquid layer. However this does not stop the Neural Pipeline architecture being useful. The work presented here is a proof of concept and the same architecture will be able to do more complex processes within the layers, that will not be achievable with a single layer. The individual layers can provide task specialisation, so that the pipeline can recognise more inputs than a single layer. Additionally it is possible that the architecture may allow the use of fewer neurons to perform the same tasks as a single layer, because of this specialisation. Further experimentation will be needed to show whether this is the case. Currently the architecture offers the advantage of providing the outputs from each layer in sequence, after a delay, rather than at once as would be the case with a single layer. Additionally the Neural Pipeline offers the computational throughput advantage provided by a traditional computer pipeline.

It has been shown that a system trained with a noiseless input responds differently to different types of input noise. It is seen that the system responds correctly for synchronised inputs even when the rate of the spikes is very different to the training set (in the Gaussian noise experiment). It is found that providing unsynchronised inputs causes the system to respond incorrectly, with even small variations causing incorrect responses. This demonstrates that the synchronisation of inputs is more important for a correct response than the precise spike timing. With synchronisation the system is robust to fluctuations in timing.

It has also been seen that increasing the amount of noise tends to gradually reduce the number of correct responses that the system gives. This result matches the hypothesis introduced in section 6.1. The result is important because the more similar an input is the the desired one, the more easily the system should recognise it. If the input differs too much from the original input then the system should not recognise it, because this increases the risk of misclassification.

6.6.1 Further work

One of the most informative extensions to the work would be to continue adding layers onto the system to test for how many layers the information provided only to layer 1 remains distinct. This is an important factor for the Neural Pipeline to be used in real world applications. The experiments (sections 6.3 and 6.4) show that information is still recognisable after three layers, but the limit still needs to be tested. This limit may be adjusted by changing the parameter settings of the Neural Pipeline, and this is also an avenue for further investigation. When the experiments are extended to include more layers, a useful extension would be to include different properties of the input pattern. For the image examples explored here colour, scale and rotation are possibilities.

The ideas proposed in the discussion above for splitting the inputs between the layers, with specialisation in later layers, to show that this can be achieved. Additionally a comparison with a single layer for computational efficiency would demonstrate the advantages of splitting the input patterns between layers. The use of different parameters to aid in specialisation should also be tested. The use of dynamic synapses should be considered in all layers, as Maass et al [49] find that correct responses are found more frequently with dynamic rather than static synapses.

More complete capacity tests for a full scale system should be carried out to test the memory limits. The factors that influence this should also be investigated, these are likely to include the separation of the inputs, the learning method used and the size of the layers. It may be that with the choice of the same number of fully connected readout neurons as system inputs, that there is no real limit imposed by the size of the layer (before the simulator's limit or the computer system limit). The limit is then going to be imposed by the time that it takes to train and run the system. Another probable outcome is that splitting n inputs between two layers, so that each layer recognises $n/2$ of the inputs, will take less time to train and run than a single larger layer. This leads to another optimisation between the number of layers and the number of inputs to be identified by each layer, because it is believed that the data will degrade over time. Too many layers will be problematic, but too few will mean that the system takes a long time to train.

One parameter that may influence capacity and duration of signal is the choice of window that the system is trained on. There are so many parameters in the system that could be varied, and time constraints meant that it was not possible to test every one of these. The methods for choosing windows and window size were fixed throughout the experiments. As illustrated in the example in figure 6.2, the choice of window size can be important in finding a unique pattern for each shape. It suggests that there is a trade off between splitting into very small windows for a larger chance of a unique set of windows, or using larger window sizes so that the process of finding a window takes less time. Also the smaller the window size, the shorter the time that the system identifies the shape for. This may not be a problem, for example, if the readouts are trained to keep spiking, but it is something that should be investigated. It is possible that smaller windows could be advantageous, because they may allow the system to respond more quickly to a particular shape. This may improve the potential throughput of the system, because the next stimulus will be able to be presented more quickly.

The method of choosing the window itself could be optimised to enhance separation. Rather than picking the first unique set of windows, all of the possible windows could be searched through to find the set with the largest amount of separation between all of the inputs. The search through all of the windows would take time to perform, but when using offline learning it only needs to be done once before training the system. If the system had to learn new inputs online then this extra step would be costly.

Generalisation should be introduced to the system via noise tolerance and position invariance. As the system trained on a perfect input is shown to be susceptible to noise, a useful extension if the working system is likely to encounter noise would be to include it in the training. This could take a number of forms, either using noisy inputs when the system is initially trained, altering the training method that is used or the readout system. If various noisy examples were presented to the system for each of the inputs, instead of a single perfect example, then the robustness to noise would be likely to increase. The use of a different and more robust training method could improve the noise tolerance of the system. An example to test is the P-delta rule [7] which leaves a margin around zero when training. This margin means that small changes to the input do not cause the sign of the response to change.

The readout mechanism used for these experiments is not biologically realistic, it was chosen here as the simplest method of displaying which input was presented. It is a filter which displays which of the inputs is being presented in a high level form using a single neuron per input. This form of readout is shown to be suitable when the input has low levels of noise, but a different readout mechanism may provide a more robust response when noise is introduced. Instead it would be possible to have a group of neurons to represent each of the inputs and use some form of averaging to determine which output is selected. Maass et al use such a system in [49]. This would mean that even if some of the readouts were altered by the noise, if the majority were correct then the system would still respond correctly. An alternative is to have a bank of neurons that match the input grid. Using the grid the readout neurons could be trained to display the original input and a human observer can recognise the input pattern from this.

In the noise experiments (in section 6.5) where examples have been given they are used to provide insight to the influence of noise. They do not represent a way of analysing exactly what type of influence a particular level of noise will have. They represent a suggestion of what can happen if different types of noise are applied. For a more in depth analysis of the influence of noise, these cases should be repeated in the same manner as the Gaussian noise experiment (section 6.5.3). In addition to this, different types of noise could be investigated including internal noise present in the system rather than superimposed onto the input.

The positional information provided to the system in this experiment was restricted to only five locations on the input grid. This is useful to show that different properties can be extracted, but with this small example the system is probably learning each of the 10 inputs as a separate image. To make the position information independent, the experiment could be extended so that the system can identify an image at any possible position on the grid. The system should be trained with only the 3 by 3 pixel shape image, rather than the entire grid.

6.6.2 Summary

A Neural Pipeline can be used as an architecture for pattern recognition. Information presented as an input to the first layer can be identified by each of the layers in the system. The layers can be trained to specialise for particular features of the input, so the recognition task can be split. When noise is introduced into the system input the system fails gradually as the noise level is increased, so that some inputs are classified correctly with low levels of noise. If the inputs are synchronised then there is inherent noise tolerance to fluctuations in timing, with the training that has been used.

The following chapter provides an overview and discussion of the work that has been done in this thesis. It describes how the Neural Pipeline architecture has been developed and tested, and summarises the main findings of the work.

Chapter 7

Conclusions and Further Work

7.1 Introduction

This chapter provides an overview of the work that has been carried out in the thesis. The Neural Pipeline architecture is described as a multi-layered neural network, with external feedforward connections that are excitatory and inhibitory feedback connections. The progress that has been made in developing and testing the architecture is summarised.

The Neural Pipeline architecture is evaluated against the objectives set out for the work in section 1.2.1. It is found that the objectives for the thesis have been achieved. In achieving the objectives the aim of the thesis has also been met. The Neural Pipeline illustrates that it is possible for a spiking neural memory to control its own data flow.

Further work suggested in the previous chapters is summarised. It is grouped into work relating to: testing the parameters, extending the architecture, training the system and analysis. Some of the suggestions are requirements to show that the architecture is usable, others are more fanciful possibilities.

7.2 Overview

This section provides an overview of the work that has been achieved in the thesis.

7.2.1 Architecture

The Neural Pipeline architecture is proposed in chapter 3, as a computational architecture that can control the timing of its own data flow, using the data itself. This inherent timing is more suited to the biologically inspired artificial neural network than alternative timing methods such as a regular clock pulse. The Neural Pipeline architecture fulfils the thesis aim of using coordinated firing to control the timing of the system.

The Neural Pipeline uses two sets of ‘external’ connections between the layers to control this data flow. There are excitatory connections providing a forward flow of data from one layer to the next. Inhibitory connections provide feedback from one layer to the previous layer to shut off the layer while processing occurs in the current layer. Through this mechanism they control when an input is allowed to pass through the system. These connections allow timing to occur in the same manner as an asynchronous pipeline with handshaking between layers to pass on the data. It is these external connections that are necessary for the Neural Pipeline architecture.

The inhibition to the last layer is provided using the system input, because there is no 'next layer' to inhibit it. In this work it is necessary to inhibit the last layer because it uses the same parameter settings as the other layers, and needs to be shut off between inputs. The inhibition to this layer is provided from the input rather than an alternative layer to allow flexibility. It means that if the delay on the connection is longer than the time for the data to pass through the entire pipeline then the input will shut off its future self in the last layer. If the delay is set to be shorter than the amount of time that it takes to pass through the architecture, then it will stop the previous signal. This allows a signal to remain active in the last layer until another input is presented.

In this thesis the layers are composed of randomly connected leaky integrate and fire neurons. They are not constrained to this type of network and could theoretically be any artificial neural network. The use of any type of neural network layer means that the architecture is a general structure. The only required elements of the architecture are the sets of external connections.

7.2.2 Behaviour

While simulating a Neural Pipeline architecture three different types of behaviour have been identified and defined. The type of behaviour that is exhibited is determined by the balance between activity in a layer and inhibition shutting down the layer. They are named according to whether the level of inhibition is correct, outweighs the activity or is too low compared to the activity. Thus they are known as 'correctly inhibited', 'over inhibited' and 'under inhibited' respectively.

When the system is correctly inhibited each input that is presented flows through each layer of the architecture, it causes activity and is then shut off. With over inhibited behaviour, after the first input has passed through the layers, the inhibition it causes is too high and prevents the next input from passing through all of the layers. In the case of under inhibited behaviour, there is not enough inhibition between layers to sufficiently shut them off.

Correct behaviour is the desired type, with 'over' the preferable of the two undesirable types. Over is preferable because it is possible to modify the input timing in order to change the behaviour to correct . If under inhibited behaviour occurs then no further inputs can be passed through without first reseting the layers to stop the spiking. These definitions help users of the architecture to identify whether the parameter settings they are using will produce the desired response.

7.2.3 Parameter Exploration

The parameters of a Neural Pipeline are split into two types; the external parameters and the internal parameters. The external parameters are those belonging to the connections that run between the layers. The internal parameters are those within the layers. The external connections have weight, delay and connectivity parameters. The internal parameters are the number of neurons and the connection parameters of weight, delay and connectivity. There are additional parameters related to the input, these are the size of the input (how many neurons it is presented to) and the duration between inputs. Finally, there are restrictions that can be applied to the choice of random connections within a layer. If a neuron is permitted to have only excitatory or inhibitory connections then it follows Dale's principle, otherwise it does not.

These parameters are investigated in chapter 4 to identify their influence on the behaviour of the system. Through the experiments it is found that the key to correct behaviour is to achieve a balance of the spiking activity within the layers and the external inhibition. Three main parameters are found to influence this balance, through experimentation. These parameters are the external inhibitory weight, the internal connectivity and the size of the input. The internal connectivity and size of input control the amount of spiking within the layers and the external inhibition is controlled by its weight. It is not necessary to use a specific value for any of these parameters, but is important to adjust the others in response when one is set. So a high value of internal connectivity or a high input size will require a high external inhibitory weight value.

In addition to this balance there are certain general choices that will make correct behaviour more likely. These choices are a low internal connectivity value and a low internal weight value. Having a low weight value increases the number of runs that exhibit correct behaviour, because the activity within the layers is reduced so less external inhibition is required to shut off the layer. A low value of connectivity produces higher numbers of correct simulation runs. Layers with a larger number of neurons are able to have a higher connectivity value and still produce correct runs. There is a limit for the connectivity value, even as the layer size is increased, and this limit is imposed by the external inhibition. As the connectivity is increased the external inhibition must be decreased to prevent over inhibited behaviour. With low connectivity values the choice of inhibitory weight is not as important, this emphasises the required balance between the parameters.

The choice of connection topology for the external connections is also important. Throughout the thesis the inhibitory connections have been fully connected, fewer connections could be used but the weights would need increasing on the remaining connections to allow the same behaviour. The excitatory connection topology is important for preserving the inputs as they are transferred along the pipeline. If the connections compress the data, for example providing input to only one neuron on the next layer, then the different inputs look the same in the second layer. If, on the other hand, different randomised connections are used from each neuron then the pattern can be preserved.

The required system input provides constraints for some of the parameters. To have a larger (uncompressed) input, a higher number of neurons are required per layer. As the size of the input is increased, the activity in the layer will increase so the external inhibition should be reduced accordingly.

A different parameter, relating to the structure of the randomised connections is whether the neurons follow Dale's principle. Dale's principle states that any particular neuron must only have connections of one type, either excitatory or inhibitory, but not both. The architecture is found to operate correctly either using Dale's principle or not, though there are some differences. The simulations that use Dale's principle have fewer parameter settings that achieve all correct runs, but retain some correct runs at more extreme parameter settings. This suggests that the system using Dale's principle could be more robust to parameter settings depending on the randomised settings of the run.

The key findings are that the Neural Pipeline architecture can operate correctly within a wide range of parameter values. The parameters can be tuned so that if one must be a particular value, the others can be adjusted to produce correct behaviour. The only system constraint for the parameters is to balance the spiking activity within the layers with the external inhibition. This illustrates that the architecture can be set up using different values depending on the desired task.

7.2.4 Analysis

Analysis of the system is given in chapter 5. The first part of analysis work concerns the causes of the three different behaviour types. It is the randomised connectivity within the layers that allows simulation runs with the same parameter settings to exhibit different types of behaviour.

As the connectivity value is equal for each neuron, this means that every neuron in the network has the same number of outgoing internal connections. They do not, however, have to have the same number of incoming connections, because the targets are chosen at random. It is found that balancing the number of excitatory and inhibitory inputs that a neuron has is more important than restricting the number of connections that it has. Having a low standard deviation for this sum of excitatory and inhibitory connections is a suggested way of encouraging correct behaviour.

The weights used for the internal connections are set either to plus or minus the same value (e.g. 2.5) or randomly within a range of values (0 to 2.5 for excitatory and -2.5 to 0 for inhibitory). When randomised weight values are used, this represents another difference in the network that contributes to determining the behaviour types. It is found that when there are neurons with more extreme values on their internal inputs, the behaviour appears to be more likely to be incorrect. Reducing the standard deviation of the input weights is therefore suggested as a way of increasing the likelihood of a correct run. Having extreme values on just one of the layers appears to be enough to cause incorrect behaviour.

The complex nature of the architecture means that it is not possible to guarantee correct behaviour by having a low standard deviation of input balance or weights. Neither will every set of values with a high standard deviation give incorrect behaviour.

The second part of analysis looks at the preservation of information through the architecture. Using a metric it is shown that information is preserved through a five layer Neural Pipeline. A variant of the metric can be used to identify how similar two inputs are and how the difference compares over time in each subsequent layer. The metric shows that the two inputs which share fewer input neurons remain distinct for longer than the two inputs which share more input neurons. The metric can be used to determine which inputs should be learnt by which layer.

7.2.5 Learning Application

The architecture has been tested on simple examples of image recognition using the principle of Liquid State Machines. Each layer can represent a single LSM because it is a randomly connected group of neurons. A separate set of readout neurons is fully connected to each of the layers, here one neuron is used per input property that is to be learnt. This was considered the simplest way for the system to display its output. Training takes place by altering the weights only on the connections between the layer and the readout. Training is performed using the delta rule for a 5ms time window of the simulation.

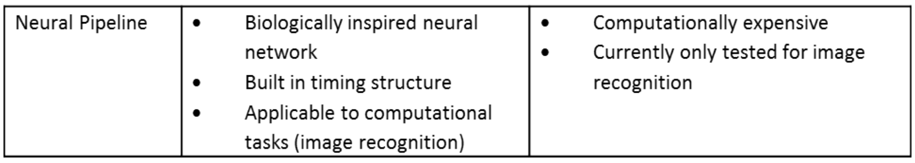
In chapter 6 it is shown that the architecture can be trained to recognise different inputs. In one experiment six different shapes can be identified, all shapes are recognised by all three layers in the architecture. This demonstrates that there is enough information present in the signals to tell the shapes apart, as identified in the analysis using the metric. A different test illustrates that it is possible to identify different properties of an input using different layers. In this test the position and the shape are the two properties that are examined, the position is identified by layer 1 and the shape by layer 2.

The impact of noise on a system trained without noise is examined, and it is found that different types of noise influence the system in different ways. With noise applied synchronously to all of the inputs, the system can perform without fault. When a large amount of noise is applied the system begins to misclassify the inputs. When the inputs have noise applied independently (rather than synchronously) the system misclassifies inputs with a much lower level of noise. This shows that the synchronisation of the spikes is more important than the actual timings themselves. The system exhibits a gradual failure as the amount of noise is increased. The conclusion from these results is that a system with a single readout neuron for each input, trained only on a perfect input, is likely to produce at least some incorrect responses when presented with an unsynchronised noisy input. There is an inherent degree of tolerance when the noise is synchronised. Including noise when training or using a more robust readout mechanism are suggestions for overcoming the sensitivity to unsynchronised noise.

These tests illustrate that the architecture is suitable for training as a LSM and is appropriate for pattern recognition tasks. It meets the aim of coordinating information in a neural network memory. The experiments show that the different layers can be used to perform different tasks, by splitting the properties of the inputs over the layers. This work provides a basis for the Neural Pipeline to be developed further and suggests ways of achieving this with future work.

7.3 Evaluation against objectives

The work completed in this thesis is evaluated according to the objectives outlined in the introduction in section 1.2.1. The fulfilment of these objectives was specified to achieve the aim outlined in section 1.2. The aim of the work was to investigate the ability of using spiking neurons to control information flow through a network



Neural Pipeline	• Biologically inspired neural network • Built in timing structure • Applicable to computational tasks (image recognition)	• Computationally expensive • Currently only tested for image recognition

Figure 7.1: The strengths and limitations of the Neural Pipeline architecture

by designing and simulating a spiking neural architecture.

Each of the objectives described in the introduction is evaluated below, with a discussion of the extent to which it has been achieved.

7.3.1 Objective 1. Compare Methods and Alternatives

The literature review in chapter 2 illustrates examples of both methods useful for achieving the aim and alternative architectures with similar functions. Chapters 3 and 4 provide justification for which of the methods from the literature were chosen for use in the architecture. Examples of choices include the neuron model and the network structure.

Various alternative architectures are examined to give an insight into the problem but also to draw inspiration from. Computer pipelines and the Synfire chain provide examples of timing architectures. Associative memories and Reservoir computers are introduced as different types of memory.

These different architectures are compared in figure 2.13 to identify how well they meet the aim outlined in this thesis. It can be seen that none of the existing architectures when used alone can meet the three key criteria required for the aim. These criteria are: being biologically inspired (neural network); applicable to computational tasks and having a timing system built into the architecture.

For comparison strengths and limitations of the Neural Pipeline architecture are outlined in figure 7.1. The strengths show that the Neural Pipeline is a novel structure which fulfils the specified aim better than any of the existing architectures. The limitations are addressed in the further work.

This objective is considered to have been met because different options were presented before a choice was made for the architecture. Existing architectures were compared and found not to meet the necessary criteria for the aim. The Neural Pipeline combines elements of LSM, Synfire Chains and asynchronous pipelines to achieve the aim.

7.3.2 Objective 2. Model and Simulate the Architecture

The architecture model is described in chapter 3, with motivation for the decisions based on the preliminary tests in chapter 4. Different parameters are tested to identify their influence on the system's behaviour, and from this the most suitable settings are chosen for the architecture generally, but also for the tests in chapter 6.

As there are so many parameters only certain ones could be investigated within the scope of the thesis. Many of the parameters are tested, with a focus on the ones considered to have the most influence. This work can be extended to include additional parameters, this is a suggestion for future work.

The architecture is simulated in its developing form in chapter 4 and in its finalised form in chapter 6. The spiking neural simulator NEST is chosen as a suitable tool for simulation.

This objective is considered complete because sufficient parameters were tested to produce a working simulation of the architecture. Chapter 3 is considered to provide an adequate description of the model so that it can be reproduced using different simulation environments.

7.3.3 Objective 3. Produce Analysis

The analysis of the system mainly relates to the three different types of behaviour that can be exhibited when the system is run. These three behaviours were identified and defined through experimentation using a simulation of a Neural Pipeline. They are defined in section 3.4. These definitions have been particularly useful when determining which parameters are most suitable. The tests in chapter 4 show how different parameters influence behaviour.

Analysis of the system is provided in chapter 5. Firstly the factors that contribute to the behaviour type seen for a particular simulation run were investigated. The behaviour of any given run depends on the exact connections of the randomly generated topology. The analysis shows that it is possible to increase the chance of having correct behaviour by having a low standard deviation for the weights on the input connections to each neuron. Balancing the number of excitatory and inhibitory connections that each neuron has is also found to have some influence. As discussed in the analysis, it is not possible to guarantee any type behaviour for any randomly generated set of connections. This is demonstrated by considering particular connection scenarios (e.g. all connections are self connections).

The second type of analysis concerns the duration that distinct inputs remain recognisable in the architecture. This is carried out using a variation of Rochel and Cohen's distance metric. It indicates that over time the inputs become less recognisable. This particularly influences later layers because the input is provided only to the first layer. The metric shows that the shapes learnt in this thesis remain distinct for three layers. The metric itself will be useful for further work when larger data sets are tested or more layers are added.

This objective has been completed because different types of analysis have been produced. Further analysis of the architecture is still possible and suggestions for this are made in the further work section.

7.3.4 Objective 4. Introduce Learning

In chapter 6 a Neural Pipeline is successfully trained to recognise different input shapes. The system is able to recognise each shape independently with each of its layers. Additional tests show that it is possible to extract different features using the different layers of the architecture.

In demonstrating that the Neural Pipeline is suitable for a simple image recognition application this objective has been achieved. The suggestions for extending the tests proposed in the further work section are necessary before the architecture can be used on more complex examples such as real world images.

7.3.5 Objective 5. Evaluate the Success Computationally and Biologically

Computationally, the Neural Pipeline architecture is considered to be successful because it fulfils the aim of being a neural network memory that can control the timing of its information by coordinating how its neurons fire. As it is a memory it can be trained to perform a useful computational task, here image recognition. The examples here are simple and demonstrate that it is possible to perform this type of task rather than aiming to solve a complex problem.

The architecture is applicable more generally to classification because different types of input can be encoded as a spike train. The architecture itself does not consider the shapes presented in chapter 6 as images but as spike trains. This means that the architecture can be applied to other types of recognition, such as sound. This is backed up by the use of LSM for different classification tasks in the literature and the fact that LSM have been shown to be universally computational.

From the experiments, although there is some inherent noise tolerance, the system could not always recognise noisy inputs. This is not a fundamental limitation of the architecture, but a limitation of the training carried out and the readout mechanism. Here, only perfect inputs were used to train the system, but noisy examples could be used. A a set of readout neurons could be used instead of the single readout neuron.

The architecture has certain limitations. One such limitation is that it is not possible, given a set of parameters, to guarantee correct behaviour. This is because the behaviour depends on the topology of the connections. This is not a large limitation because it is easy to detect whether the system behaves correctly and to choose a new set of parameters if it does not. Additionally, there are ways to increase the chances of achieving correct behaviour. Minimising the number of individual neurons with particularly high input weights is one method. The choice of reasonable parameters, such as low connectivity, is another way to encourage correct behaviour.

Another limitation is the need to choose parameters for a task. So for example to perform the experiments in chapter 6, a layer size of 100 neurons was required along with a reduced external inhibition. This limitation is considered to be acceptable, because it is not expected that such a system can handle wide variations in input without any change. Chapter 4 outlines suggestions for parameter choice, such as a low connectivity. The experiments presented in this chapter suggest that the system is able to produce correct behaviour for a wide range of values. This means that the parameter choice does not need to be 'perfect' to make a working system.

To evaluate the system biologically is more difficult. Through the study no matching architecture has been identified in the biological literature. The Synfire Chain is believed to be biologically plausible and it is a similar architecture. The Synfire Chain is, however, less biologically restrictive than the Neural Pipeline in terms of its connection structure. It is feedforward and has no interconnections within layers.

While it is not possible to say that such an architecture could be found in a brain, the use of biologically sensible components (spiking neuron models) and connectivity (Dale's principle), along with the observation of Pieron's Law, mean that a similar architecture could be possible.

7.4 Further Work

There are many suggestions, from the work that has been done, to extend the architecture further. These ideas have been introduced in the discussion section of each chapter (sections 3.5.1, 4.9.1, 5.4.1 and 6.6.1). These different possibilities can be grouped into the following categories: continued parameter exploration, extensions to the architecture, improvements to learning and further analysis of the architecture.

7.4.1 Parameter Exploration

During the development of the architecture, only certain parameters have been investigated. The remaining parameters have been fixed at only one or a few values for the experiments. In order to explore how the architecture responds to changes in these parameters, they could be investigated in the same way as those presented here, by fixing the other values and varying the test parameter. To perform these tests for every parameter would be time consuming. Instead it may be possible to work out how the most of the parameters contribute to the overall layer activity. From the experiments already conducted, it has been found that it is the balance of the activity in the layers with the external inhibition, that controls the behaviour of the system. If an equation could be produced using these parameters then an estimate of their influence on the layer activity could be found, thus indicating their influence on the behaviour. The internal weight values, external excitatory weight, input size and neuron parameters could be combined to produce an estimate of the activity.

Some of the parameters, such as the internal and external delays, describe the timing of the activity rather than the the level of activity. It may therefore be best to investigate these parameters separately, as with the experiments presented here. The choice of whether to put a delay on the external inhibition or the external excitation is an important factor to investigate. This choice impacts the length of time that the layer can operate for before the next layer is started, so dictates the timing of the outputs from the readout neurons. The best length of delay for a particular task should also be considered, some tasks may require more time for the layers to calculate than other tasks. Investigating the length of delay includes the longer delay to the last layer. In this work this delay has been set to be longer than it takes for the data to pass through all of the layers. It is suggested that with a shorter delay, an input would be able to stop the result of the previous input, instead of its own signal. This needs to be tested to see whether it provides an advantage. Allowing the last layer to continue to spike may mean that the information contained in the activity degrades, so there may not be an advantage when the inputs are presented with a large gap between them.

7.4.2 Adjustments to the Architecture

Several possible changes are proposed that could be made to the architecture. It is believed that some of these changes will make the architecture more specialised for a particular task. Other changes, such as changing neuron type, test that the system will work under different conditions to those tested in the thesis. They should retain the same operation that the architecture currently has.

The experiments here all use LIF neurons but the architecture should be able to use any type of spiking neuron. The system is trained in the same way as LSMs, which traditionally use LIF neurons, but LSMs have successfully been constructed from Hodgkin-Huxley neurons [39]. This suggests that the Neural Pipeline should also be able to use Hodgkin-Huxley neurons.

Internally the layers follow Dale's principle, but the external connections do not. There are two possible ways to apply Dale's principle to the external connections. Firstly just to remove the connections that do not follow Dale's principle and secondly to insert circuits of inter-neurons to correct the connections that violate Dale's principle. Both of these alternatives should be compared to the existing architecture, for correct behaviour and computational expense. The best of the three should then be used as part of the architecture.

The architecture has been developed so that it can have any number of layers, but has only been tested with fairly small numbers, most frequently three or five. It is important to test the number of layers that can be used while still being able to recognise different inputs. It may be the case that parameter changes can be made that will improve this number, for example a smaller external excitatory weight and corresponding reduction in layer activity may help. The benefit of introducing additional layers is that the different layers can be trained to recognise different inputs or different properties of the inputs. It is necessary to identify how many layers are suitable for different numbers of inputs.

The suggestions of alternative forms of the architecture could be tested as future work. One is suggested to handle continuous input in a different fashion to the standard Neural Pipeline architecture. They should be compared when presented with a continuous input stream to identify which is preferable. The second alternative allows a different signal to control the timing of the system, this architecture would also allow the same operation as the traditional architecture by using the same input for both timing and processing. This still fulfils the objective of a self coordinated system, because the second signal is just a trigger to begin the timing from within the system.

A later possibility for future work is to implement the architecture in hardware. Once extensive testing is completed using the simulation it may be desirable to have an implementation of the architecture that can respond more quickly. One of the current limitations of the architecture is the speed of the simulation. Producing a hardware version of the system, for example on an FPGA, would be a way of addressing this. It may also be possible to produce the architecture using real neurons in order to gain further insight into the biological plausibility of the system.

7.4.3 Extensions to Learning

The examples of training the network presented in chapter 6 are simple examples to test that the architecture is able to perform image recognition. The set of shapes used is small and they are noiseless. This means that the system is not able to generalise or to cope with large amounts of input noise. If the architecture is to be used for more realistic examples of image recognition then these are necessary extensions.

The method of training the system used in the experiments presented here does not train the system to generalise. A useful extension would be to introduce generalisation so that the system could recognise inputs that are presented in any position, rather than just specified locations. The system should also be trained to deal with noisy inputs. This should improve the ability to deal with different types of noise and with larger amounts of noise. It should be achievable because LSM are trained to produce the correct response while using noisy inputs in [49]. It may require a more robust readout system and different training. Internal noise could be introduced in addition to noise on the input.

The training in the examples presented here uses ‘time windows’, so the readout neurons recognise an input at a particular point in time. The window size is not investigated here, but may be useful when learning larger numbers of inputs. The choice of window size will cause a trade off between the length of time it takes to train and the number of possible patterns that can be learnt. The choice of which window to use is an area that could allow improvement in telling apart the different inputs, and also in generalising. Currently the first unique set of windows is chosen, with no consideration as to how similar the windows are for different inputs. It would be possible to choose the most dissimilar windows, so that it is easier to identify different inputs. To help with generalising, windows could be taken at a time when the inputs to be classified as the same shape are more similar.

There are possibilities for introducing different types of input, for example with shape, position, orientation, scale and colour. Once simple examples of these are tested, the system can be trained to recognise real world images. Another consideration is applying a continuous input such as a stream of images or a video, and how the splitting of the signal by the layer will influence this. Testing the application of a continuous input should help to identify the limitations of the architecture.

7.4.4 Analysis of the Architecture

Further analysis of the system is the final area for future work that is discussed here. There are several areas that have been identified that provide areas for analysis.

The system behaviour represents a possible area for analysis. By considering the sum of input weights to each neuron from within the layer, it is suggested that where there are extreme values incorrect behaviour is more likely. These extreme values increase the standard deviation of input weight. It may be possible to identify a value of standard deviation below which a certain percentage (say 95%) of runs will be correct. It is thought that although this may help with the understanding of behaviour, it would be simpler to generate a randomised set of connections and test the behaviour rather than trying to produce sets that are correct using the analysis.

Two hypotheses are presented to try to explain the change from correct, to over inhibited, to under inhibited behaviour seen when the connectivity of the layers is increased. These hypotheses should be tested to identify if either are correct in their explanation. This may not be as useful as the extensions for learning or exploring the parameters of the architecture, but may help to provide an understanding of the behaviour.

The idea that the system may be able to produce under inhibited behaviour between certain layers and over inhibited between others is another consideration. If it can be identified under which conditions this type of behaviour occurs then it will aid with understanding whether all layers of the architecture have to have particular settings for the system as a whole to behave correctly. The standard deviation of the weights on the inputs to the neurons may be a way of analysing each of the layers independently.

The capacity of the architecture, when used for learning, is a useful area for analysis. The capacity will be altered depending on the system parameters, from the preliminary tests presented in this work it is suggested that the layer size is important. The training method and the difference between the inputs is also likely to have an influence. The number of different inputs (or input properties) that can be stored by a layer is important when choosing parameter values for the architecture. It is suggested that by splitting the inputs that are to be recognised between layers, the system could be made more efficient because it will be easier to train. The signal does appear to degrade as it passes through the layers, so this will need to be addressed. This could be achieved using feedback from the readout neurons in the earlier layers or alternatively by classifying the inputs with least separation with the earlier layers.

7.5 Summary and Key findings

The Neural Pipeline demonstrates that a neural network memory architecture can produce internal timing using structured connections. The novel architecture shows that it is possible to use handshaking principles to control the behaviour of a layered neural network. The excitatory feedforward connections between each layer of the architecture provide the input to the following layer. Inhibitory feedback connections are used to prevent data transfer while the layer is busy processing.

Three types of behaviour have been identified and investigated. The behaviour type is determined by the strength of the inhibition compared to the level of activity within the layer. The balance can be correct or can be over or under inhibited. When in the correct region, each input produces a wave of activity that propagates through each of the layers. The activity within the layers is controlled mainly by the connectivity and input size, this is balanced with the external inhibition to encourage correct behaviour.

As an application example it is demonstrated that a Neural Pipeline can be used for image recognition. Different system layers can be trained to perform different computational tasks, here identifying shape and position. There is some inherent noise tolerance which could be improved by training for noise.

The Neural Pipeline architecture has the potential to be further developed for use in more complex problems, such as computer vision. The analysis of how distinct input signals remain with time will be useful as the data sets and number of layers are increased. The Neural Pipeline is a general architecture. Each of its layers is a Liquid State Machine which is universally computational so a Neural Pipeline should be applicable to any computational task.

Appendices

Appendix A

Appendix

The appendix contains additional information, figures and tables to supplement the work.

A.1 Dale's Principle on the External Connections

Dale's Principle (introduced in section 2.2.3) is used for the internal connections within the layers, this is described in section 4.6.3. The external connectivity does not, however, follow Dale's Principle because all of the neurons, including the excitatory ones, have inhibitory connections. This is not an issue, because although Dale's Principle was used to make the system more biologically sensible, it is not necessary for the architecture to operate correctly. This is shown by the results presented in section 4.6.3.

It is also possible to make the system follow Dale's Principle, with certain alterations. For the the feedback this could be achieved using inhibitory inter-neurons between each of the excitatory neurons in layer *n+1* and the neurons in layer *n*. This is shown in figure A.1 (a). The inhibitory neurons can connect directly to the neurons in layer *n* as they do currently. Each excitatory neuron in layer *n+1* has its own inhibitory inter-neuron which is fully connected to the neurons in layer *n*. This allows each of the neurons to have outgoing connections only of their own type, thus following Dale's Principle.

This method would not work on the feedforward connections, because an excitatory inter-neuron cannot be made to spike using an inhibitory connection. This is shown in figure A.1 (b). Alternatives would be to only connect the excitatory neurons to the next layer. This may reduce the amount of spiking seen in later layers, compared with the current architecture, because fewer spikes are being passed forward. This need not be a problem as the weight on the external excitatory connections actually increases the number of spikes in subsequent layers as discussed in section 4.6.3. It could even improve the region of correct behaviour shown in figures 4.11 and 4.12, by delaying the onset of over inhibited behaviour as the connectivity is increased. This is consistent with hypothesis 2 presented in section 4.6.3. It is a possibility to be investigated in further work (section 3.5.1), to see if the system still works correctly and whether this change can improve the behaviour as suggested.

Another alternative would be to replace the connection with a neural circuit that is capable of changing a negative input to a positive output. An example of this is shown in figure A.1 (c). This clearly adds more overhead to the system than only using connections from the excitatory connections or breaking Dale's principle. The spike generator is used to provide a continuous input spike train. The overhead is not trivial because this set of neurons would be required for every one of the excitatory external connections from an inhibitory neuron. This option will also have the possibly unwanted side effect of synchronising the timing of the signals with the spike generator, as the circuit is of the same form as the ‘synchronisation module’ introduced by Maass in [45]. In this case, as the circuit is being used for a different purpose, the synchronisation of the signals with the spike generator may lose signal information. The results for introducing input noise to the system in section 6.5 suggest that this may not be an issue, if the system is trained with the synchronised signal. These complications mean that Dale’s Principle is not used for the external connections in this thesis.

The last of these solutions (with individual neural circuits for each connection) is not biologically likely because of the large overheads and specific connection structure required. As Dale’s Principle is introduced to improve the biological plausibility this option appears to be a poorer choice. It is preferable either to remove the connections which contravene Dale’s Principle or to let the external connections contravene Dale’s Principle. The choice of which of these options is the most suitable should be addressed through further work and is discussed in section 3.5.1.

A.2 Alternative Layouts

There are possible alternatives for the structure, which still follow the same principle of data progressing using excitatory connections and being controlled using inhibitory ones. These possibilities have not been tested, as the decision was made to focus on setting up and testing the original architecture initially.

Two suggestions are shown in figure A.2, in (a) the stages are separated into computational stages C and buffer stages B. The computational stages each perform a required task and the buffer stages provide a mechanism for producing output at a given time. Not inhibiting the computational stages means that they can continuously compute rather than being completely stopped between inputs. This would allow a continuous input to be sampled in a different way to the traditional Neural Pipeline architecture (see section 4.9.1 for suggestions of continuous inputs to a Neural Pipeline). It would not be suitable for the stream of separated inputs that is presented to the Neural Pipeline, because it would always exhibit under inhibited behaviour as the computational stages are not inhibited. This means that the structure of architecture should be chosen based on the required input.

Architecture (b) in figure figure A.2 uses a Neural Pipeline architecture to provide the timing for a set of computational processes. This allows a second input to control the timing of the process. This still fulfils the requirement of achieving timing using the system data, because an external input is not required. It does not restrict the architecture to using its own input for timing, as the existing Neural Pipeline architecture does.

Either of these architectures may be useful depending on the requirements. Other possibilities for variations also exist and could be created for specific tasks. These variations form an area for future work and are discussed in section 3.5.1.

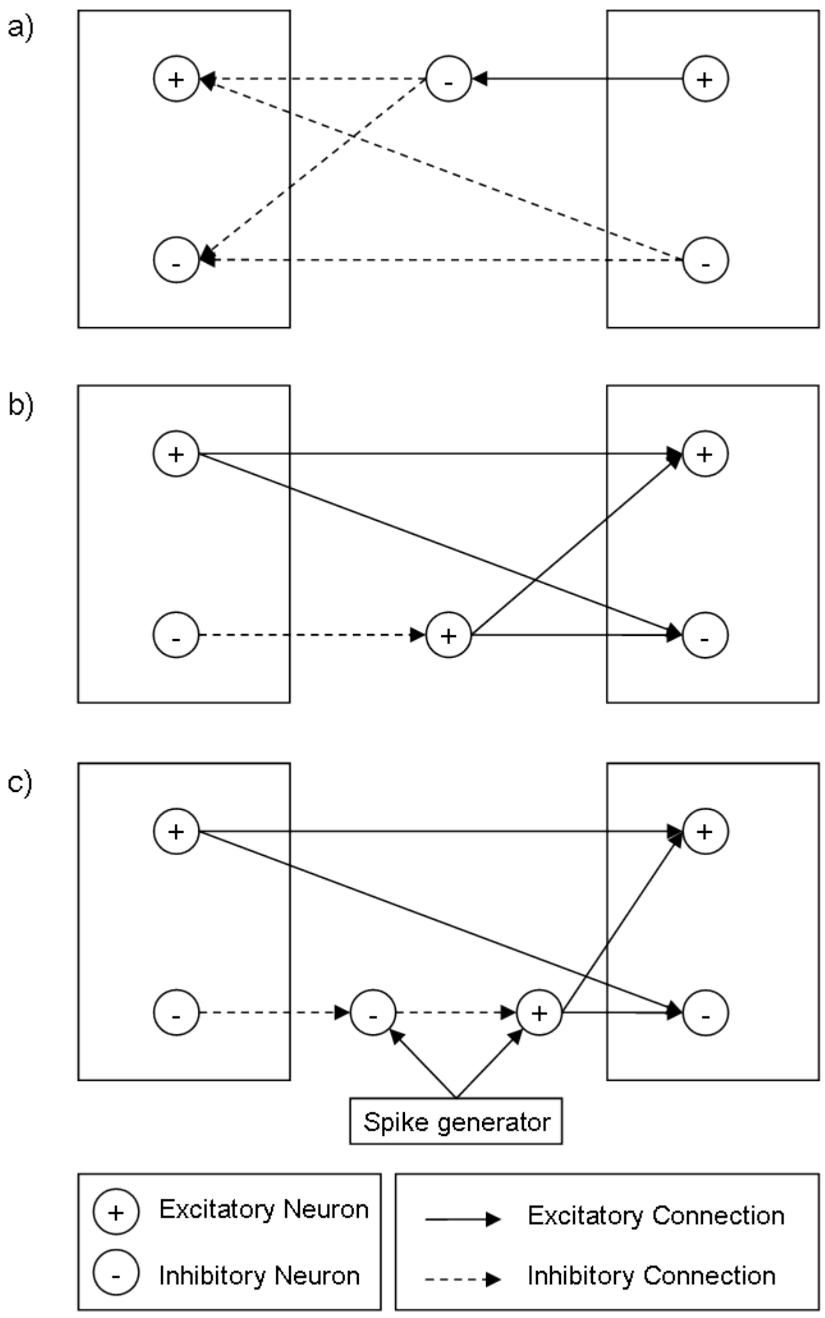


Figure A.1: The use of inter-neurons to make the external connections follow Dale's Principle. (a) shows the feedback connections, and that the inter-neuron can make the system follow Dale's Principle. (b) Shows why this will not work for the feedforward connection, because the excitatory inter-neuron cannot be made to spike using an inhibitory input. c) Shows an alternative circuit of inter-neurons that would work for the feedforward connections from inhibitory neurons.

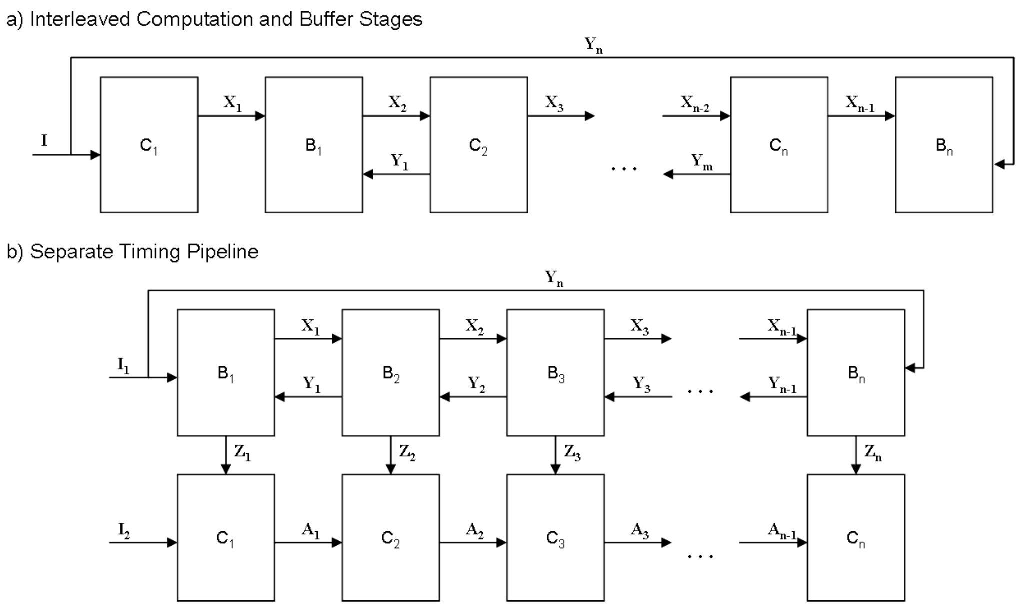


Figure A.2: Two alternative architectures. (a) has computational layers that do not receive inhibition, followed by the standard pipeline layers. These layers act as buffers to delay the signal. (b) Has a separate timing pipeline to control the architecture that performs the task.

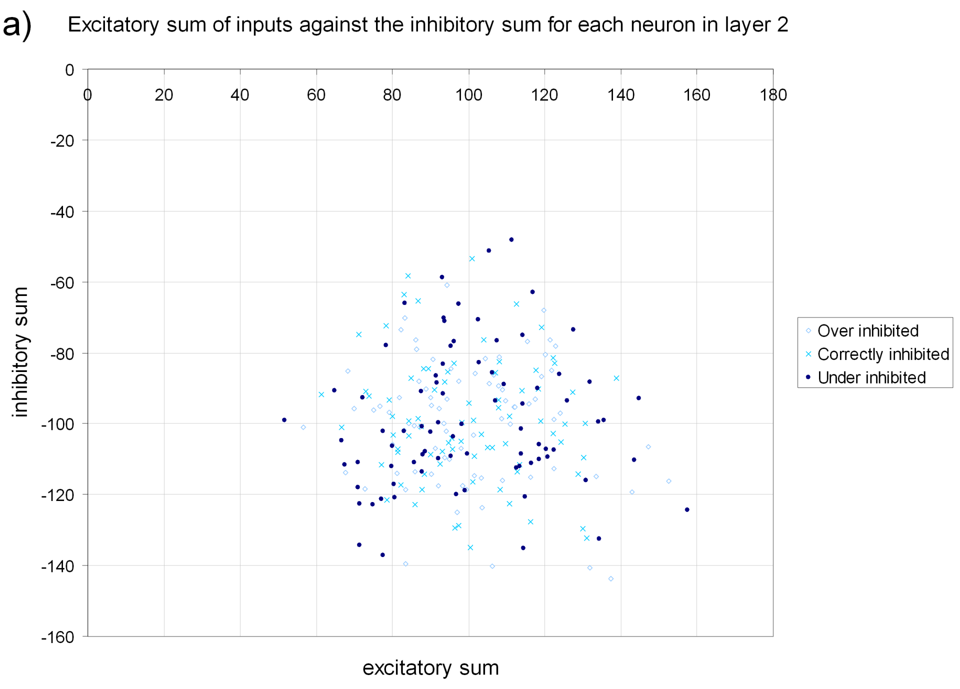
A.3 State Space

The Neural Pipeline has a large state space, because at any given time any of its neurons can fire or not fire. The experiments presented in chapter 6 do not require this space to be investigated, because it is so large when compared to the number of shapes that are stored. It is important that different inputs should take a different trajectory through the space so that they can be identified. As the network is randomly connected and different inputs are defined by being connected to different neurons it is expected that this true. Liquid state machines work on this principle (see section 2.11.1 for an overview of LSM). The trajectory that the different inputs take has not been investigated here, because they have been found to be sufficiently different for learning in the experiments presented in chapter 6. They do represent a useful area for future work, because it is possible that it could be used as a method of generalising if similar inputs tend towards similar trajectories.

A.4 Analysis

The F-Test is used in section 5.2.2 to compare two sets of data. It is used to determine how probable it is that both data sets have the same variance. The null hypothesis is that the two variances are the same. A one tailed test is performed to test whether the correct data has a lower variance than either of the incorrect runs. The p value returned by the test shows how likely it is that this result would be found by chance assuming that the two variances are the same.

a)



b)

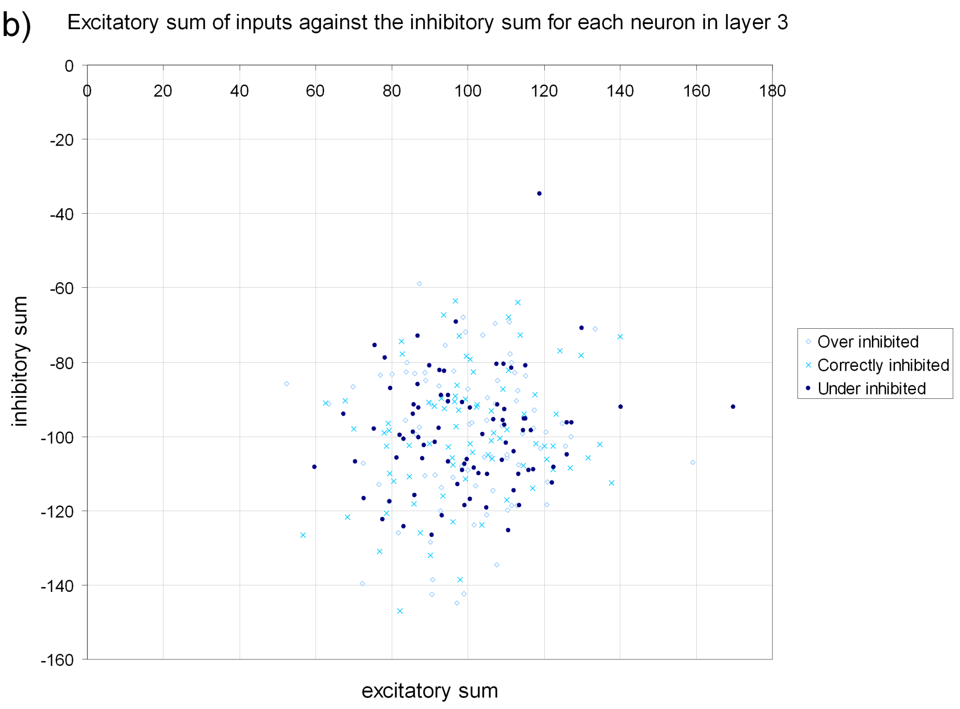
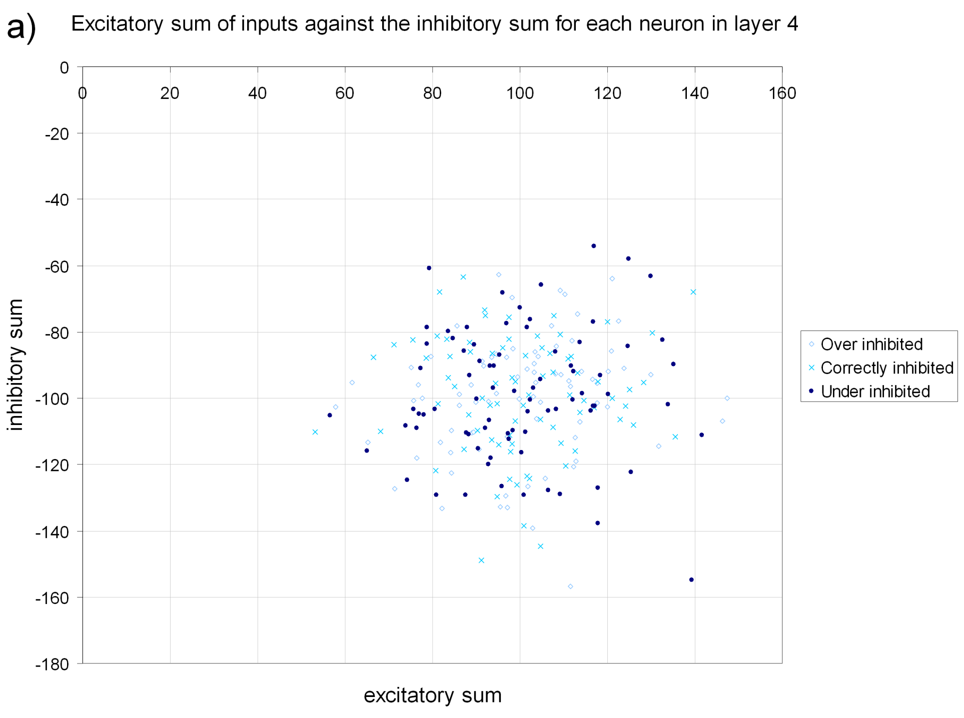


Figure A.3: The sum of the excitatory weights that are on the input connections plotted against the sum of inhibitory connections. The plots show a point for all neurons for an example of each type of behaviour. Layer 2 is shown in plot (a) and layer 3 in plot (b).



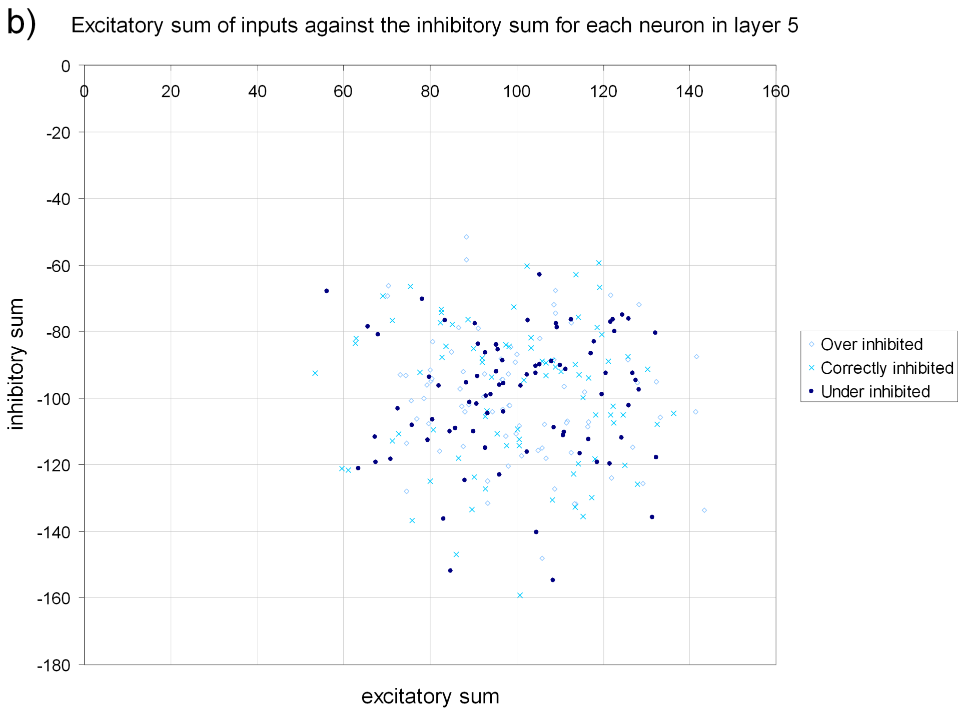


Figure A.4: The sum of the excitatory weights that are on the input connections plotted against the sum of inhibitory connections. The plots show a point for all neurons for an example of each type of behaviour. Layer 4 is shown in plot (a) and layer 5 in plot (b).

	Layer	correct < over	correct < under
Excitatory Weights	1	0.0907	0.0256
	2	0.2188	0.0378
	3	0.6682	0.4587
	4	0.2593	0.1868
	5	0.7528	0.6343
Inhibitory Weights	1	0.0347	0.0594
	2	0.5494	0.1554
	3	0.1593	0.8634
	4	0.4346	0.1796
	5	0.8923	0.8746

Table A.1: The p value found using the f-test with a null hypothesis that the variances of the weight values are the same for the different behaviours.

Layer	Behaviour	excitatory st dev	inhibitory st dev
B1	Over	18.36749	18.09698
	Correct	15.80634	14.75267
	Under	19.68606	17.58079
B2	Over	19.04961	17.13632
	Correct	17.46023	17.37653
	Under	21.32417	19.47103
B3	Over	16.806	19.27073
	Correct	17.64629	17.22947
	Under	17.85296	15.2348
B4	Over	17.26477	17.88932
	Correct	16.05858	17.56199
	Under	17.74548	19.46617
B5	Over	17.94465	18.65316
	Correct	19.37446	21.43778
	Under	18.64251	18.84346
Mean	Over	17.8865	18.2093
	Correct	17.26918	17.67169
	Under	19.05024	18.11925

Table A.2: Standard deviations of the input connections to each neuron, for all layers for one run of each type of behaviour.

	Layer	correct $<$ over	correct $<$ under
Excitatory Weights	1	0.7393	0.6066
	2	0.5586	0.9741
	3	0.1046	0.5484
	4	0.9375	0.6648
	5	0.8721	0.5903
Inhibitory Weights	1	0.9679	0.3491
	2	0.8302	0.6418
	3	0.0399	0.5180
	4	0.7540	0.9116
	5	0.7521	0.5876

Table A.3: The p value found using the f-test with a null hypothesis that the variances of the number of connections are the same for the different behaviours.

	Layer	correct $<$ over	correct $<$ under
80 neurons	1	0.2894	0.2227
	2	0.6318	0.6154
	3	0.1120	0.3599
	4	0.7001	0.8349
	5	0.4302	0.3147
70 neurons	1	0.6249	0.8307
	2	0.3710	0.2439
	3	0.2201	0.1570
	4	0.2057	0.8326
	5	0.7159	0.9566
60 neurons	1	0.0669	0.0624
	2	0.5219	0.3631
	3	0.9403	0.8254
	4	0.9181	0.5354
	5	0.1909	0.7660

Table A.4: The p value found using the f-test with a null hypothesis that the variances of the sum of the weights is the same for the different behaviours. This test is performed for three different layer sizes.

A.4.1 Distance Metric

A.4.1.1 Parameters

The parameters used here are different from those used in [56] because they are chosen to produce correct behaviour in the Neural Pipeline architecture. The reason for this is because the experiment is performed to examine the behaviour of the architecture with parameters that it is likely to use rather than to replicate the experiment in [56] using multiple layers.

A.4.1.2 Window Size

The chosen size of window can alter the results that are seen using the metric. In [56] the size is chosen 'to be sufficiently large to ensure our distance metric is viable yet sufficiently small that all or nearly all neurons have fired at most once'. The impact of window size on the results is shown in figure A.5 for four different window sizes for which almost all neurons fire at most once. The larger sizes of window (c and d) average the activity, so that the large peak that can be seen in the size 10 case (a) is barely seen. The small window sizes mean that a single neuron can have a larger impact on the overall distance. So the window should be large while still fitting the criterion of almost all neurons spiking at most once per window.

With the Neural Pipeline there is the additional difficulty of keeping the layers comparable, because the number of spikes seen in each layer varies considerably. To keep the layers comparable with each other it is sensible to use the same size of window. With the subjective time measure the choice of window must be small, because there are few spikes on the first layer. If the window size is bigger than the number of spikes then layer 1 has no comparison data. With the real time version it is possible to have larger windows, because all layers are run for the same duration. However larger windows can result in multiple spikes per window for some of the neurons in the later layers.

A.4.1.3 Presenting specific shape inputs

When the subjective time metric is used to compare the shapes, the response is different to the result found in the architecture with a random stimulus. The graphs are shown in figure A.6. The two inputs remain different for around the first 10 spikes on each layer. On layers 2 and 3 the difference then oscillates between a large and a small difference in signals. This is because a small sliding window is used. The oscillations occur as the influence of a particular neuron slides into or out of the window. The window size in figure A.6 is 10 spikes per window dictated by layer 1 which was observed to have as few as 13 spikes. Layer 2 and 3 have more spikes so can have larger windows, the graphs are shown in figure A.7. The behaviour is still oscillatory, but the oscillations are smaller and are smoothed. These results are an artefact of the metric rather than a property of the Neural Pipeline.

	Variable name	Value	Description
experiment parameters	total_stimuli	10	Number of different input stimuli
	total_runs	10	Number of different internal setups
	sim_time	100	The duration of the simulation in ms
	window_size	50	The size of the window used for the distance metric
layer parameters	num_buffs	5	The number of layers in the pipeline
	num_neurons	100	The number of neurons in each layer
	stimulus_size	5	The number of neurons per layer to receive an input stimulus
connection parameters	ex_ex_weight	5.0	The external excitatory weight value
	ex_ex_delay	1.0	The external excitatory delay value
	in_weight	-2.0	The external inhibitory weight value
	in_delay	5.0	The external inhibitory delay value
	ex_weight	0 to 5.0	The internal excitatory weight value range
	int_in_weight	0 to -5.0	The internal inhibitory weight value range
	no_conns	(num_neurons/2)	The number of excitatory connections from each neuron
	no_conns	(num_neurons/2)	The number of inhibitory connections from each neuron

Table A.5: The parameters used for the distance metric tests performed in the analysis.

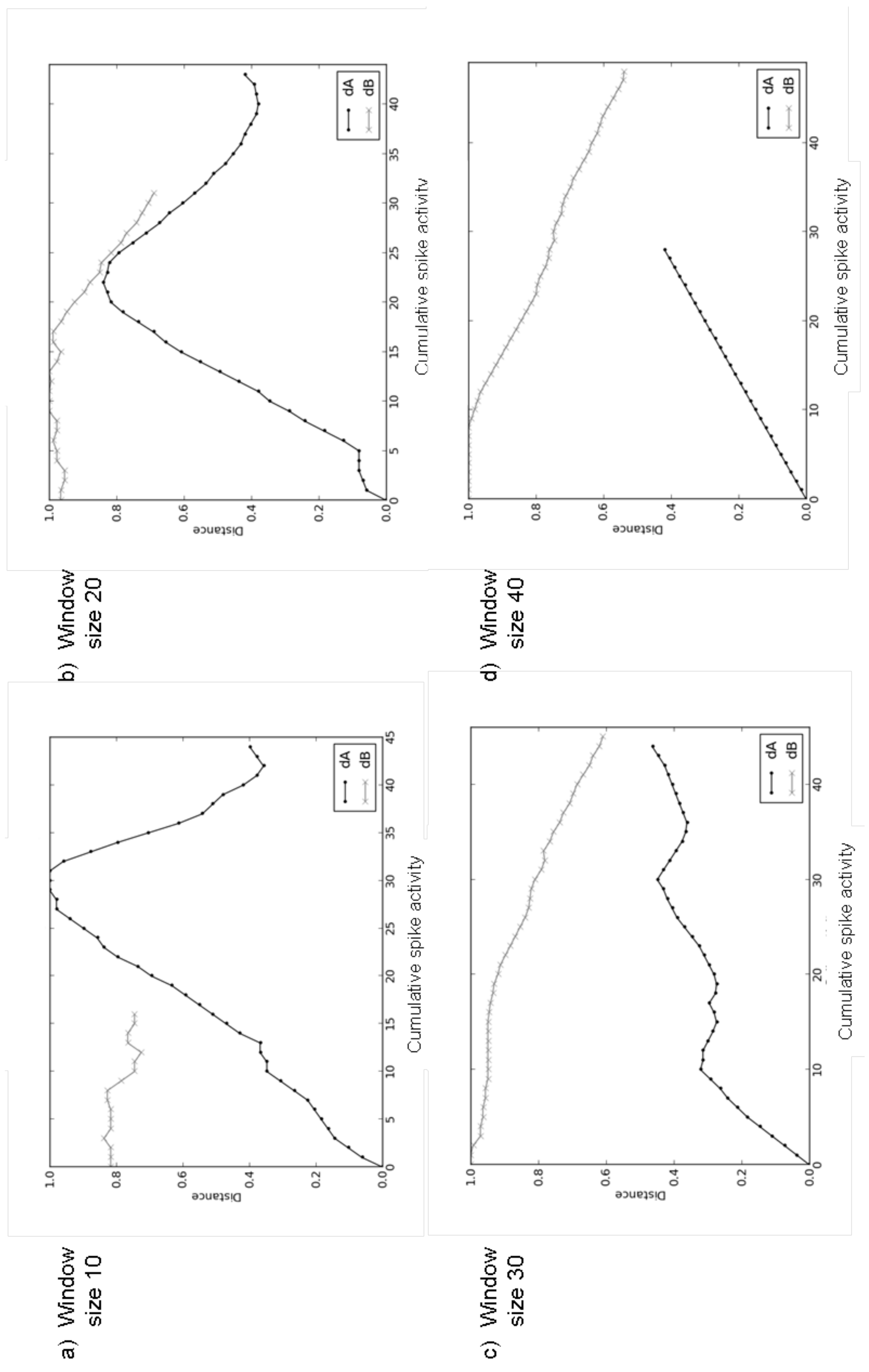


Figure A.5: The average distance over 5 internal settings and 5 different inputs for each of 4 different window sizes.

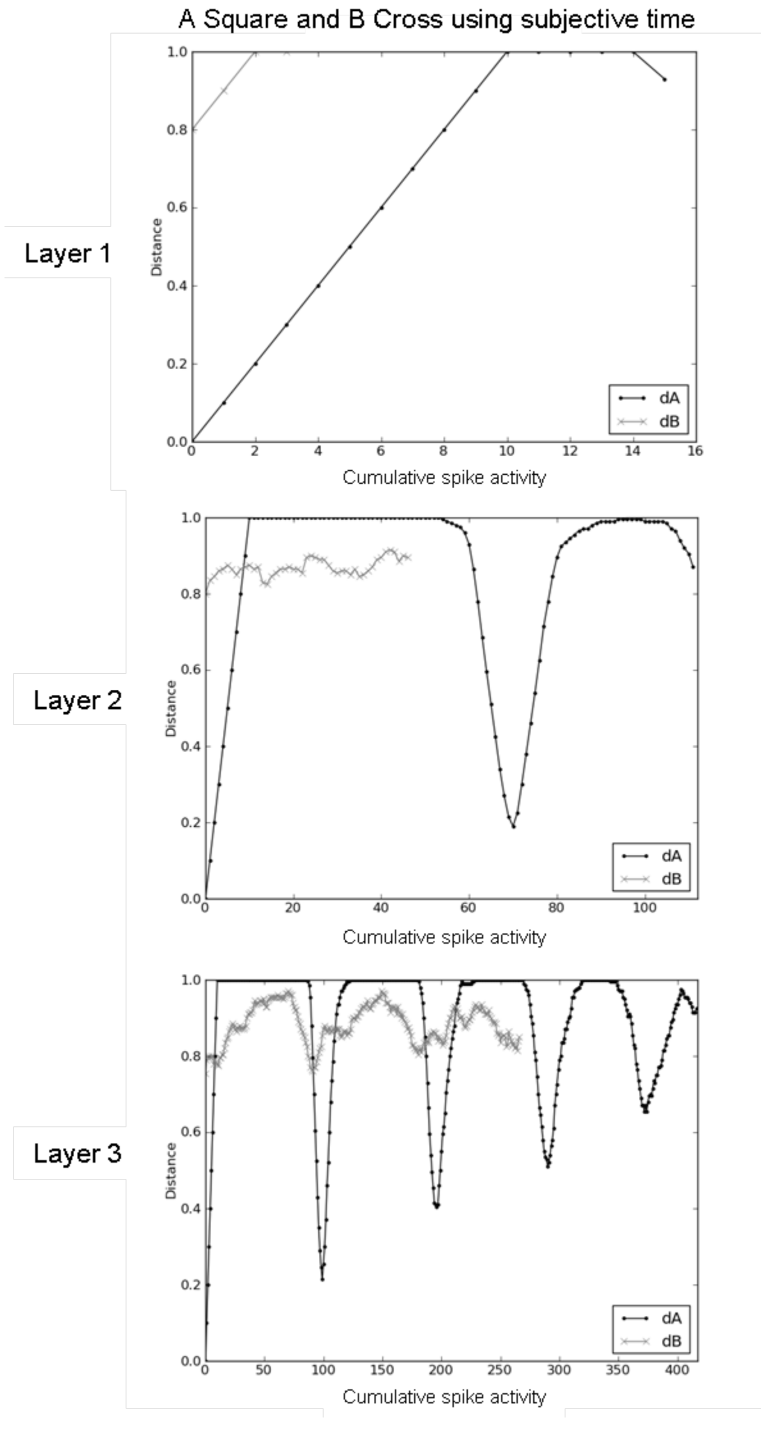
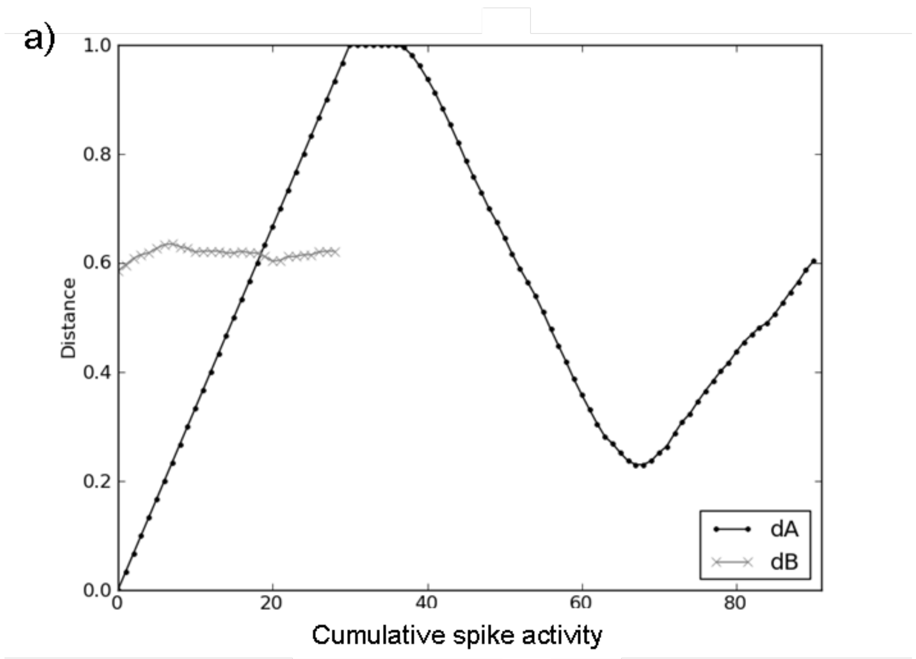


Figure A.6: The distance metric when applied to all layers of a Neural Pipeline. Square and Cross are the two inputs being compared, using a cumulative spike total for a 'subjective' time measure.



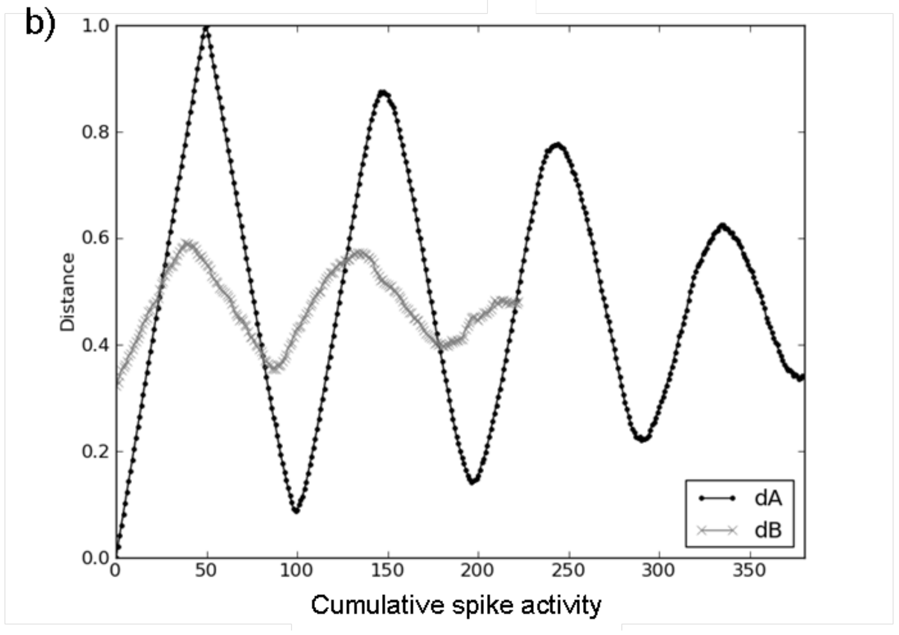


Figure A.7: The distance metric comparing square and cross for a) layer 2 with window size 30 and b) layer 3 with window size 50.

A.5 Parameter tests

The transition between correct and over inhibited behaviour is shown in figure A.8. An internal weight value of 0.5 (graph b) shows an unusual transition between the two behaviour types. There are two peaks rather than the smooth change experienced for all of the other experiments. This smooth change is shown for weights of 0.49 (graph c) and 0.51 (graph a). This suggests that there is something particular about a weight of 0.5 that causes this strange transition. It is believed to be a symptom of the simulation itself rather than something fundamental to the architecture.

Figures A.9 and A.10 display the tests for Pieron's law when the number of layers and the number of neurons in a layer are varied. It can be seen that the graph shapes are consistent as these values are changed.

The full correctness values and numbers of perfect runs for the graphs in figures 6.21 to 6.26 are provided in table A.6. These values are provided for a complete comparison, but the overall trends are illustrated by the graphs.

A.6 Learning as a LSM

The parameters used for the learning experiments presented in chapter 6 are given in table A.7 to allow the experiments to be reproduced.

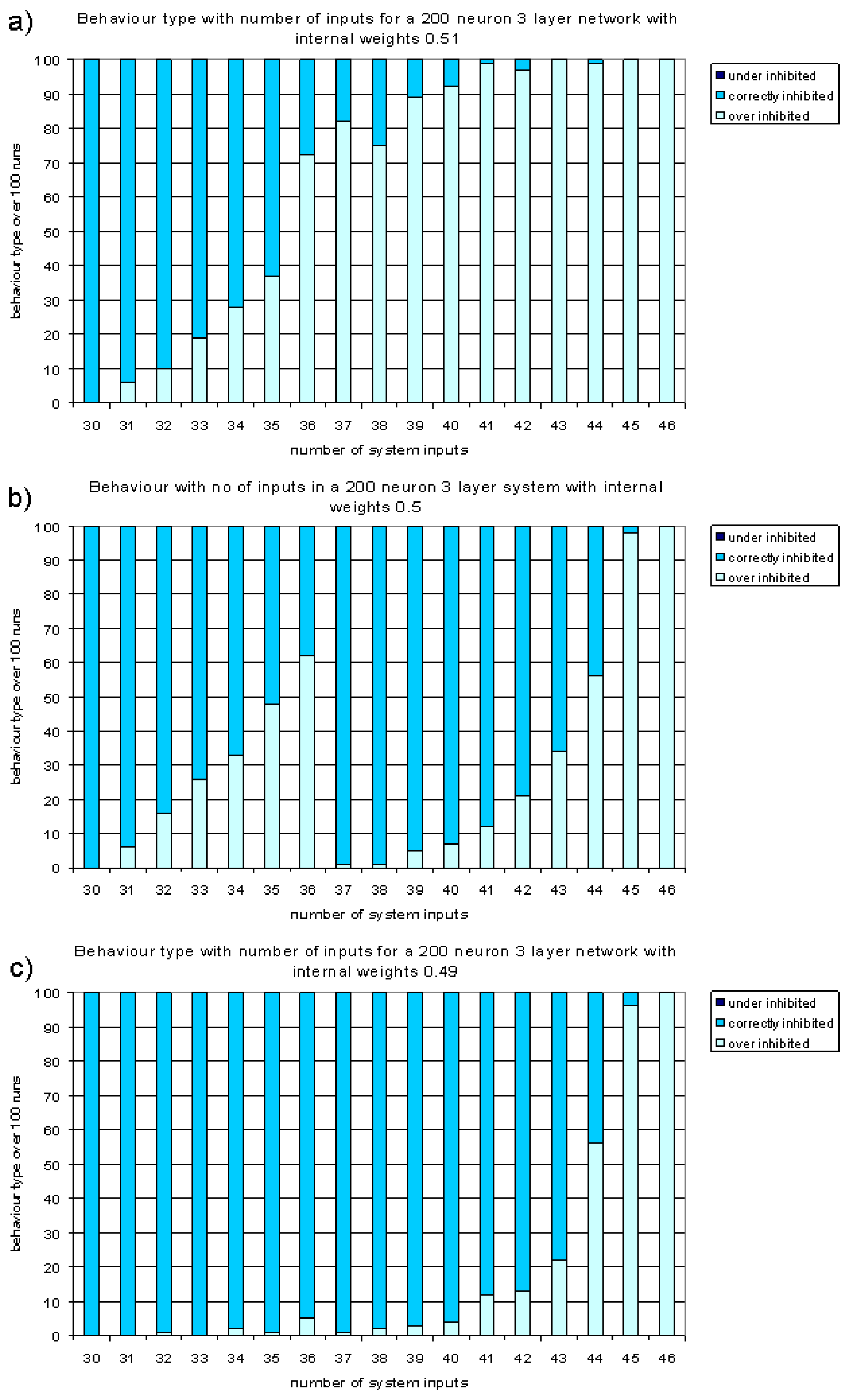


Figure A.8: Graphs showing the strange transition between correct and over inhibited behaviour with internal weights 0.5 (graph b). Graphs (a) and (c) show that weights 0.01 either side of these do not exhibit this type of transition.

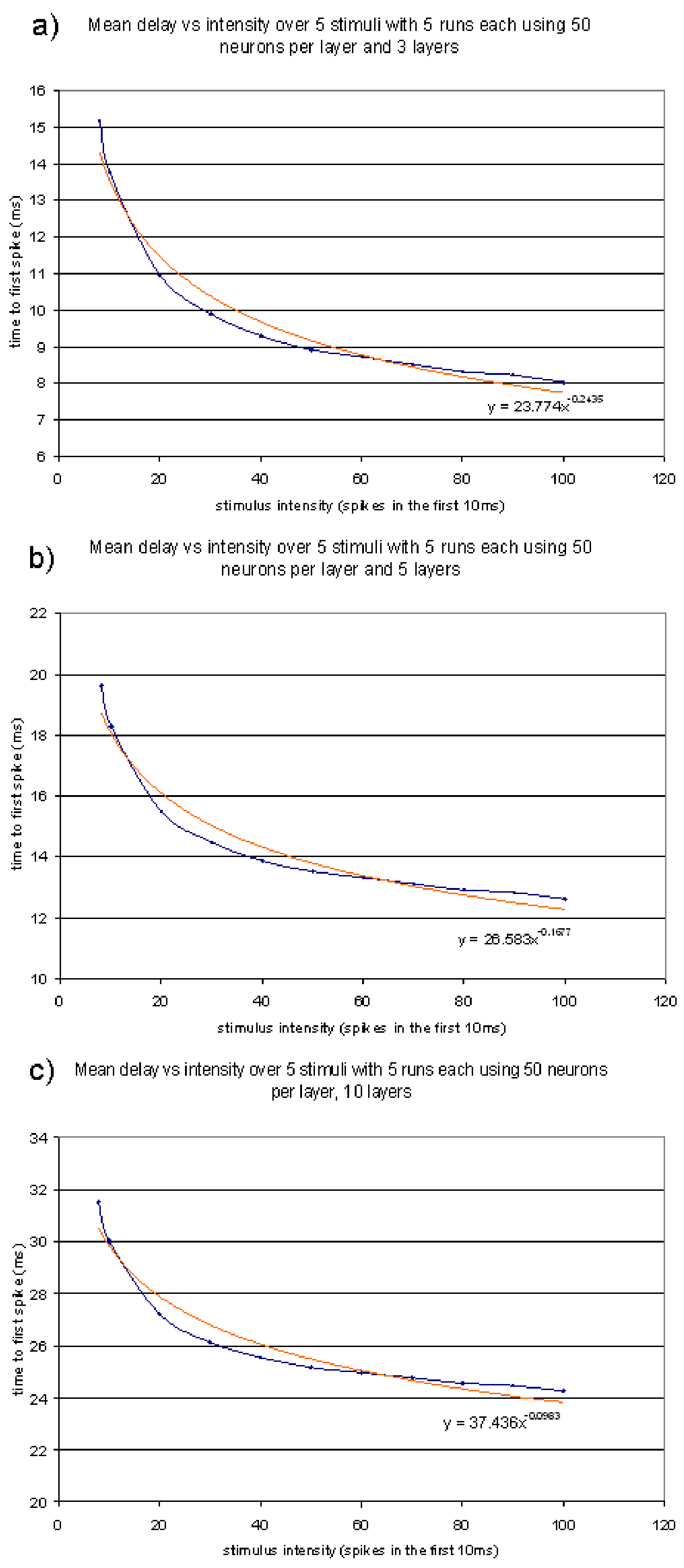
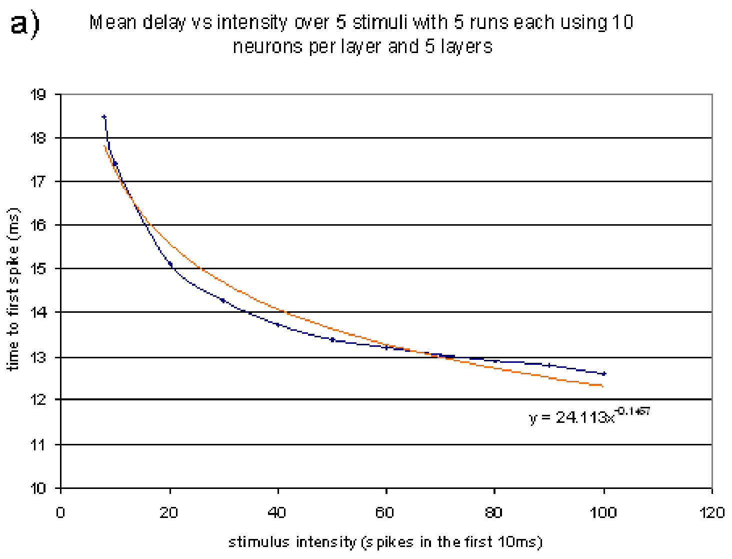
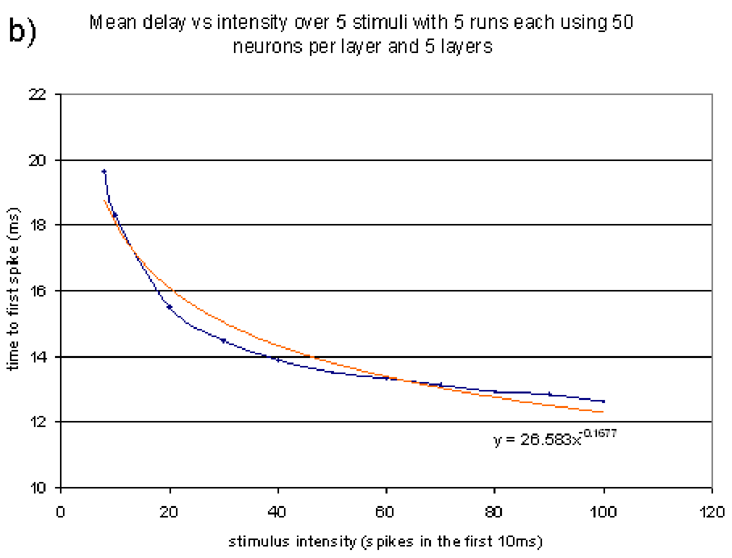


Figure A.9: The mean response time with different sizes of stimulus over 5 runs. The graphs show the data and a best fit line. Graph (a) shows the result with 3 layers, (b) has 5 layers and (c) 10.

a)



b)



c)

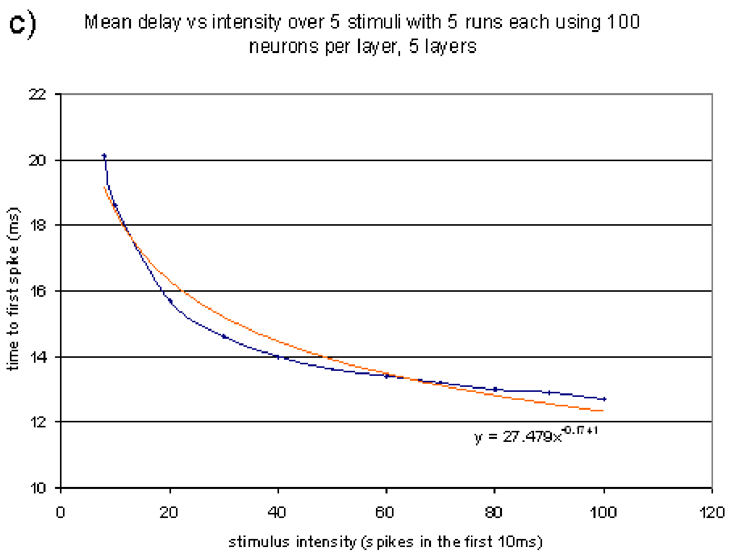


Figure A.10: The mean response time with different sizes of stimulus over 5 runs. The graphs show the data and a best fit line. Graph (a) shows the result with 10 neurons per layer, (b) has 50 neurons and (c) 100.

Input	Training	Figure	Correct Val			Perfect Runs		
			B1	B2	B3	B1	B2	B3
Poisson rate 1/ms	Regular	6.21	3	4	3	4	3	4
Poisson rate 1/ms	Regular	6.21	2	2	2	4	3	3
Poisson rate 1.4/ms	Regular	6.21	-2	4	0	0	0	3
Poisson rate 1.4/ms	Regular	6.21	-2	4	1	1	0	3
Poisson rate 1/ms	Poisson	6.22	1	3	2	1	4	3
Poisson rate 1/ms	Poisson	6.22	0	2	3	0	3	4
Poisson rate 1.4/ms	Poisson	6.22	6	3	-2	6	4	2
Poisson rate 1.4/ms	Poisson	6.22	3	3	0	4	4	3
Poisson Reg Comb	Regular	6.24	2	4	3	2	4	4
Poisson Reg Comb	Regular	6.24	1	3	1	1	4	3
Poisson Reg Comb	Poisson	6.24	0	3	3	0	4	3
Poisson Reg Comb	Poisson	6.24	0	2	2	0	2	3
Poisson to n/2 inputs (rest reg)	Regular	6.25	0	6	2	3	3	3
Poisson to n/4 inputs (rest reg)	Regular	6.25	2	6	3	4	3	4
Poisson to n/8 inputs (rest reg)	Regular	6.25	5	4	1	5	5	2
Poisson to n/10 inputs (rest reg)	Regular	6.25	6	4	1	6	5	3
Poisson to n/12 inputs (rest reg)	Regular	6.25	6	4	1	6	5	3
Poisson to n/2 inputs (rest poisson)	Regular	6.26	2	6	-1	4	4	2
Poisson to n/4 inputs (rest poisson)	Regular	6.26	5	2	1	5	4	3
Poisson to n/8 inputs (rest poisson)	Regular	6.26	5	5	-1	5	5	2
Poisson to n/10 inputs (rest poisson)	Regular	6.26	5	6	0	5	6	3
Poisson to n/12 inputs (rest poisson)	Regular	6.26	5	6	0	5	6	3

Table A.6: The correctness values and number of perfect responses for each example graph in figures 6.21 to 6.26.

	Variable name	Value	Description
layer parameters	num_buffs	3	The number of layers in the pipeline
	num_neurons	100	The number of neurons in each layer
	stimulus_size	81	The number of neurons per layer that can receive an input stimulus
connection parameters	ex_ex_weight	5.0	The external excitatory weight value
	ex_ex_delay	1.0	The external excitatory delay value
	num_ex_ex_conns	10	The number of excitatory connections from each neuron in one layer to neurons in the next
	in_delay	5.0	The external inhibitory delay value
	in_weight	-0.3	The external inhibitory weight value
	ex_weight	0.5	The internal excitatory weight value
	int_in_weight	-0.5	The internal inhibitory weight value
	no_conns	10	The number of connections from each neuron

Table A.7: The parameters used for the learning experiments.

Bibliography

[1] H. D. Abarbanel, M. I. Rabinovich, A. Selverston, M. V. Bazhenov, R. Huerta, M. M. Sushchik, and L. L. Rubchinskii. Synchronisation in neural networks. *Physics-Uspekhi*, 39(4):337 – 362, 1996. (Cited on page 26.)

[2] L F Abbott. Learning in neural network memories. *Network*, 1:105–122, 1990. (Cited on page 27.)

[3] M. Abeles. *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press, 1991. (Cited on page 48.)

[4] D. J. Amit, K. Y. M. Wong, and C. Campbell. Perceptron learning with sign-constrained weights. *Journal of Physics A Mathematical General*, 22:2039–2045, June 1989. (Cited on page 28.)

[5] Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. Storing infinite numbers of patterns in a spin-glass model of neural networks. *Phys. Rev. Lett.*, 55:1530–1533, Sep 1985. (Cited on page 41.)

[6] Hans-Martin R. Arnoldi, Karl-Hans Englmeier, and Wilfried Brauer. Translation-invariant pattern recognition based on Synfire chains. *Biological Cybernetics*, 80:433–447, 1999. (Cited on page 49.)

[7] Peter Auer, Harald Burgsteiner, and Wolfgang Maass. Reducing Communication for Distributed Learning in Neural Networks. In *Artificial Neural Networks ICANN 2002*, volume 2415 of *Lecture Notes in Computer Science*, pages 133–133. Springer Berlin / Heidelberg, 2002. (Cited on pages 127 and 166.)

[8] Jim Austin. Distributed Associative Memories for High Speed Symbolic Reasoning. *Fuzzy Sets Syst.*, 82(2):223–233, 1996. (Cited on pages 42 and 44.)

[9] Ben A. Barres. The Mystery and Magic of Glia: A Perspective on Their Roles in Health and Disease. *Neuron*, 60:430 – 440, 2008. (Cited on page 26.)

[10] Azam Beg, P. W. Chandana Prasad, and Ajmal Beg. Applicability of feed-forward and recurrent neural networks to boolean function complexity modeling. *Expert Syst. Appl.*, 34(4):2436–2443, 2008. (Cited on page 39.)

[11] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, November 1995. (Cited on page 31.)

[12] Wassila Boukhari and Mohamed Benyettou. Personal Palmprints Based-Identification using Liquid State Machine. In *ACIT 2008*, 2008. (Cited on pages 51, 125, 126 and 133.)

[13] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James Bower, Markus Diesmann, Abigail Morrison, Philip Goodman, Frederick Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23:349–398, 2007. (Cited on page 70.)

[14] Grant Brewer. *Spiking Cellular Associative Neural Networks for Pattern Recognition*. PhD thesis, University of York, Computer Science Department, 2008. (Cited on pages 43 and 44.)

[15] Nicolas Brodu. Quantifying the Effect of Learning on Recurrent Spiking Neurons. In *IJCNN 2007.*, pages 512 – 517, august 2007. (Cited on pages 52 and 132.)

[16] Dmitry Bryliuk and Valery Starovoitov. Application of Recirculation Neural Network and Principal Component Analysis for Face Recognition. In *The 2nd International Conference on Neural Networks and Artificial Intelligence*, pages 136–142, 2001. (Cited on page 39.)

[17] J. Buhmann and K. Schulten. Influence of Noise on the Function of a 'Physiological' Neural Network. *Biological Cybernetics*, 56(5):313–327, July 1987. (Cited on page 28.)

[18] Giovani Dematos, Milton S. Boyd, Bahman Kermanshahi, Nowrouz Kohzadi, and Iebeling Kaastra. Feedforward Versus Recurrent Neural Networks for Forecasting Monthly Japanese Yen Exchange Rates. *Asia-Pacific Financial Markets*, 3(1):59–75, February 1996. (Cited on page 39.)

[19] M. Diesmann, M.-O. Gewaltig, and A. Aertsen. Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402:529–533, 1999. (Cited on page 49.)

[20] John Eccles. From Electrical to Chemical Transmission in the Central Nervous System. *Notes and Records of the Royal Society of London*, 30:219–230, 1976. (Cited on pages 27 and 92.)

[21] Jochen M Eppler, Moritz Helias, Eilif Muller, Markus Diesmann, and Marc-Oliver Gewaltig. PyNEST: a convenient interface to the NEST simulator. *Frontiers in Neuroinformatics*, 2, 2009. (Cited on page 70.)

[22] Chrisantha Fernando and Sampsa Sojakka. Pattern Recognition in a Bucket. In *Advances in Artificial Life*, volume 2801 of *Lecture Notes In Computer Science*, chapter 63, pages 588–597. Springer Berlin / Heidelberg, 2003. (Cited on pages 51 and 127.)

[23] W. Gerstner and W.M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, Cambridge, 2002. (Cited on pages 9 and 33.)

[24] Marc-Oliver Gewaltig and Markus Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007. (Cited on pages 60, 69 and 70.)

[25] Christopher Glaze and Todd Troyer. Temporal variability in a synfire chain model of birdsong. *BMC Neuroscience*, 9(Suppl 1):P28, 2008. (Cited on page 49.)

[26] Dan F M Goodman and Romain Brette. The Brian simulator. *Frontiers in Neuroscience*, 3, 2009. (Cited on page 70.)

[27] Bruce Graham and David Willshaw. Capacity and information efficiency of the associative net. *Network*, 8:35, 1997. (Cited on page 43.)

[28] Simon Haykin. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, July 1998. (Cited on pages 9, 25, 27, 28, 31, 35, 37, 42 and 60.)

[29] Robert Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, 1990. (Cited on pages 28, 40, 41 and 42.)

[30] M. L. Hines and N. T. Carnevale. The NEURON simulation environment. *Neural Comput.*, 9:1179–1209, August 1997. (Cited on page 70.)

[31] Stephen Hobson. *Correlation Matrix Memories: Improving Performance for Capacity and Generalisation*. PhD thesis, University of York, Computer Science Department, 2011. (Cited on page 43.)

[32] J. J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *PNAS*, 79(8):2554–2558, April 1982. (Cited on page 40.)

[33] Jiangshuai Huang, Yongji Wang, and Jian Huang. The Separation Property Enhancement of Liquid State Machine by Particle Swarm Optimization. In *Proceedings of the 6th International Symposium on Neural Networks: Advances in Neural Networks - Part III*, ISNN 2009, pages 67–76, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on pages 52, 87 and 132.)

[34] Kai Hwang and Faye A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, Inc., New York, NY, USA, 1st edition, 1986. (Cited on page 46.)

[35] E.M. Izhikevich. Simple Model of Spiking Neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569–1572, Nov. 2003. (Cited on pages 33 and 34.)

[36] Eugene M. Izhikevich, Niraj S. Desai, Elisabeth C. Walcott, and Frank C. Hoppensteadt. Bursts as a unit of neural information: selective communication via resonance. *Trends in Neurosciences*, 26(3):161 – 167, 2003. (Cited on page 34.)

[37] John Kennedy Jim Austin and Ken Lees. The Advanced Uncertain Reasoning Architecture, AURA. Technical report, University of York, Computer Science Department, 1995. (Cited on pages 43 and 44.)

[38] Jim Jubak. *In the Image of the Brain, breaking the barrier between the human mind and intelligent machines.* Little, Brown and Company, 1992. (Cited on pages 28 and 37.)

[39] Wieslaw A. Kaminski and Grzegorz M. Wojcik. Liquid State Machines Built of Hodgkin-Huxley Neurons Pattern Recognition and Informational Entropy. In *Annales UMCS Informatica AI, Vol. I, Lublin*, pages 107–113, 2003. (Cited on pages 51, 125, 126, 133 and 178.)

[40] Daniel Kustrin and Jim Austin. Spiking Correlation Matrix Memory, 1998. (Cited on pages 43 and 44.)

[41] Daniel Kustrin and Jim Austin. Synchronisation in Spiking CMM architectures, 1998. (Cited on pages 42 and 44.)

[42] A. Lewis, S. Mostaghim, and M. Randall. *Biologically-inspired Optimisation Methods: Parallel Algorithms, Systems and Applications.* Studies in Computational Intelligence. Springer, 2009. (Cited on page 67.)

[43] Yong Li, Zheng Tang, GuangPu Xia, and RongLong Wang. A Positively Self-Feedbacked Hopfield Neural Network Architecture for Crossbar Switching. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 52(1):200–206, Jan. 2005. (Cited on page 40.)

[44] Lyle N. Long and Guoliang Fang. A Review of Biologically Plausible Neuron Models for Spiking Neural Networks. In *AIAA InfoTech@Aerospace Conference*, April 2010. (Cited on page 32.)

[45] Wolfgang Maass. Lower Bounds for the Computational Power of Networks of Spiking Neurons. *Neural Computation*, 8:1–40, 1995. (Cited on page 186.)

[46] Wolfgang Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Networks*, 10:1659–1671, 1997. (Cited on pages 28 and 32.)

[47] Wolfgang Maass and Christopher Bishop. *Pulsed Neural Networks.* MIT Press, 1999. (Cited on page 32.)

[48] Wolfgang Maass, Robert A. Legenstein, and Henry Markram. A New Approach towards Vision Suggested by Biologically Realistic Neural Microcircuit Models. In *Proceedings of the Second International Workshop on Biologically Motivated Computer Vision*, pages 282–293, London, UK, 2002. Springer-Verlag. (Cited on pages 51, 52, 125, 126, 127 and 133.)

[49] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14(11):2531–2560, November 2002. (Cited on pages 50, 51, 52, 59, 125, 126, 127, 128, 129, 132, 136, 145, 146, 164, 165, 166 and 179.)

[50] Stephen Marsland. *Machine Learning: An Algorithmic Perspective.* CRC Press, 2009. (Cited on page 131.)

[51] David Norton and Dan Ventura. Preparing More Effective Liquid State Machines Using Hebbian Learning. In *IJCNN 2006, Int. Joint Conf. on Neural Networks*, 2006. (Cited on pages 52 and 132.)

[52] S.M. Nowick and M. Singh. High-performance asynchronous pipelines: An overview. *Design Test of Computers, IEEE*, 28(5):8 –22, sept.-oct. 2011. (Cited on pages 48 and 58.)

[53] Dan W. Patterson. *Artificial Neural Networks Theory and Applications.* Prentice Hall, 1st edition, 1996. (Cited on pages 9, 35 and 36.)

[54] Phil Picton. *Neural Networks.* Palgrave, 1994. (Cited on pages 9, 29, 30, 31, 36, 37, 38, 41, 42 and 45.)

[55] Kevin L. Priddy and Paul E. Keller. *Artificial Neural Networks: An Introduction (SPIE Tutorial Texts in Optical Engineering, Vol. TT68).* SPIE- International Society for Optical Engineering, 2005. (Cited on page 40.)

[56] Olivier Rochel and Netta Cohen. Real time computation: Zooming in on population codes. *Biosystems*, 87(2-3):260 – 266, 2007. (Cited on pages 11, 59, 113, 115, 116, 117, 122, 136 and 193.)

[57] Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65:386–407, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).). (Cited on page 30.)

[58] Gordon M. Shepherd. *Neurobiology.* Oxford University Press, 2nd edition, 1988. (Cited on pages 24, 26 and 27.)

[59] S. Shinomoto. A Cognitive and Associative Memory. *Biological Cybernetics*, 57(3):197–206, October 1987. (Cited on page 28.)

[60] Takashi Shinozaki, Hideyuki Câteau, Hidetoshi Urakubo, and Masato Okada. Controlling synfire chain by inhibitory synaptic input. *Journal of the Physical Society of Japan*, 76(4):044806, 2007. (Cited on pages 49 and 56.)

[61] Jens Spars and Steve Furber. *Principles of Asynchronous Circuit Design: A Systems Perspective.* Springer Publishing Company, Incorporated, 1st edition, 2001. (Cited on page 48.)

[62] T. Stafford and K. Gurney. The role of response mechanisms in determining reaction time performance: Pieron's law revisited. *Psychonomic Bulletin and Review*, 11:975–987, 2004. (Cited on pages 10 and 101.)

[63] Thomas Trappenberg. *Fundamentals of Computational Neuroscience.* Oxford University Press, USA, June 2002. (Cited on pages 26 and 33.)

[64] Frank van der Velde and Marc de Kamps. A neural architecture for grounded cognition: Representation, structure, dynamics and learning. *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 961–968, June 2008. (Cited on page 38.)

[65] L.R. Veloso and J.M. de Carvalho. Neural versus Syntactic Recognition of Handwritten Numerals. In *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on*, pages 233–236, Sep 1999. (Cited on page 39.)

[66] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391 – 403, 2007. Echo State Networks and Liquid State Machines. (Cited on page 50.)

[67] David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Isolated Word Recognition Using a Liquid State Machine. In *In ESANN'05, European Symposium on Artificial Neural Network*, pages 435–440, 2005. (Cited on page 51.)

[68] Alan L. Wilkes and Nicholas J. Wade. Bain on Neural Networks. *Brain and Cognition*, 33:295–305, 1997. (Cited on page 28.)

[69] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222:960–962, 1969. (Cited on page 41.)

[70] G.M. Wojcik and W.A. Kaminski. Pattern Separation in the Model of Mammalian Visual System. In *International Symposium on Parallel Computing in Electrical Engineering, 2006*, pages 309 –312, sept 2006. (Cited on pages 51, 52, 125, 126 and 133.)

[71] Grzegorz M. Wojcik and Wieslaw A. Kaminski. Liquid state machine and its separation ability as function of electrical parameters of cell. *Neurocomputing*, 70(13-15):2593 – 2597, 2007. (Cited on pages 51, 52, 125, 126 and 133.)

[72] Tadashi Yamazaki and Shigeru Tanaka. The cerebellum as a liquid state machine. *Neural Networks*, 20(3):290 – 297, 2007. (Cited on page 52.)