Plasticity in large-scale neuromorphic models of the neocortex

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering.

2016

James Courtney Knight

School of Computer Science

Contents

This thesis contains 25722 words.

List of Figures

List of Tables

List of Acronyms

AMPA is the name of a family of receptors which Glutamate – a neurotransmitter released into the synaptic cleft by excitatory neurons – binds onto causing the opening of fast-acting ion channels.

ANN Artificial Neural Networks are computational models which – inspired by the biological brain – consist of large networks of neural units and can be used to solve machine learning problems.

ASIC Application-Specific Integrated Circuits are chips customised for a particular use (unlike FPGAs which can be reprogrammed).

BCPNN Bayesian Confidence Propagation Neural Network.

CPU Central Processing Units are the electrical circuits at the heart of almost all computer systems which carries out the instructions specified by a computer program.

DMA Direct Memory Access is a feature of many computer architectures which allows hardware subsystems to access memory directly rather than going through the CPU. This can improve overall system performance by freeing up the CPU for other things.

DTCM Data Tightly-Coupled Memory is located on the processor die and therefore provides very low-latency, deterministic access to critical data.

fMRI Functional Magnetic Resonance Imaging is a non-invasive imaging technique which uses a strong magnetic field and radio waves to measure blood flow in the brain and hence detect areas of activity.

FPGA Field-Programmable Gate Arrays are chips containing a large number of programmable logic blocks whose function and the wiring connecting them can be configured after manufacturing, typically using a Hardware Description Language.

GABA is the name of both a neurotransmitter released into the synaptic cleft by inhibitory neurons and of a family of receptors that these neurotransmitters bind onto.

GPU Graphics Processing Units are specialised electronic circuits included in many types of computer which were historically designed to accelerate the rendering of 2D and 3D graphics. However, more recently, GPUs' highly-parallel architectures have been employed to accelerate a wide range of algorithms.

HPC High-Performance Computing generally refers to the distribution of large, complex problems across parallel computer systems such as clusters or supercomputers so they can be run reliably and quickly.

Hz Hertz are a unit of frequency defined as one cycle per second. In digital electronics Hz are used to measure clock speed. Additionally, in neuroscience, Hz are often used to measure the expected number of events a Poisson processes might be expected to emit every second.

ISI Interspike Intervals are the times between successive spikes.

ITCM Instruction Tightly-Coupled Memory is similar to DTCM but used to store instructions rather than data.

LIF Leaky integrate-and-fire is a simple spiking neuron model, described in more detail in section 3.1.1.

LUT Lookup tables are arrays containing the pre-calculated results of evaluating a function which can subsequently be used at runtime rather than evaluating the function.

NMDA is, like AMPA, the name of a family of receptors which Glutamate binds onto. However, unlike the ion channels which open in response to AMPA binding, the ion channels opened by NMDA binding are not only slower acting but also require a baseline level of activation.

NoC Network-on-Chip is a term used for packet-based network technologies used to connect together multiple cores, caches etc in a modern CPU (rather than using the type of busses seen in older architectures).

ODE Ordinary Differential Equations contain one of more functions of a *single* independent variable and its derivatives (unlike a Partial Differential Equation which may contain functions of multiple independent variables).

SDRAM Synchronous Dynamic Random Access Memory is a generic name for various types of volatile memory (its contents is lost when the computer is turned off) which are synchronised to the CPU clock.

STDP Spike-timing dependent plasticity is a hypothesis as to how the efficacy of connections between biological neurons (known as synapses) based on the activity of the neurons. STDP is discussed in detail in chapter 4.

TLU Threshold Logic Units were an early computation model of neurons, described in more detail in section 3.1.1.

WTA Winner-Take-All is a computational principle applied to Artificial Neural Networks by which output neurons mutually inhibit each other while recurrently exciting themselves. This results in only the output neuron with the highest activity remaining active.

Abstract

PLASTICITY IN LARGE-SCALE NEUROMORPHIC MODELS OF THE NEOCORTEX

James Courtney Knight

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2016

The neocortex is the most recently evolved part of the mammalian brain and enables the intelligent, adaptable behaviour that has allowed mammals to conquer much of planet earth. The human neocortex consists of a thin sheet of neural tissue containing approximately 20×10^9 neurons. These neurons are connected by a dense network of highly plastic synapses whose efficacy and structure constantly change in response to internal and external stimuli. Understanding exactly how we perceive the world, plan our actions and use language, using this computational substrate, is one of the grand challenges of computing research. One of the ways to address this challenge is to build and simulate neural systems, an approach neuromorphic systems such as SpiNNaker are designed to enable.

The basic computational unit of a SpiNNaker system is a general-purpose ARM processor, which allows it to be programmed to simulate a wide variety of neuron and synapse models. This flexibility is particularly valuable in the study of synaptic plasticity, which has been described using a plethora of models. In this thesis I present a new SpiNNaker synaptic plasticity implementation and, using this, develop a neocortically-inspired model of temporal sequence learning consisting of 2×10^4 neurons and 5.1×10^7 plastic synapses: the largest plastic neural network ever to be simulated on neuromorphic hardware. I then identify several problems that occur when using existing approaches to simulate such models on SpiNNaker before presenting a new, more flexible approach. This new approach not only solves many of these problems but also suggests directions for architectural improvements in future neuromorphic systems.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

Firstly I would like to thank Marli Roode. Many years ago, outside an East London pub she convinced me that I should quit my job and start a masters degrees and, over subsequent years, she has continued to inject entropy into my life.

For making this plan a reality I would like to thank my parents who have been nothing but supportive of this, somewhat drastic, change in life direction; Simon Moore who first placed the fanciful ideas of neuromorphic engineering in my head; and Steve Furber who, from our first email conversation onwards, has offered invaluable advice, support and just the correct amount of rope to hang myself in all number of interesting ways.

Since the first day I sat down next to them in IT302, Jonathan Heathcote and Andrew Mundy have been great lunch buddies, willing readers, stoic providers of Linux support and great friends to travel the world with. I would also like to thank Jim Garside for being such a welcoming force in the APT group – my PhD would have been nowhere near as much fun without his organisation of trips to various pubs and hills.

Finally I would like to thank the Engineering and Physical Sciences Research Council (EPSRC) and the President's Doctoral Scholar Award for financially supporting me during my throughout my PhD.

Chapter 1

Introduction

The neocortex is the distinguishing feature of the mammalian brain and enables mammals to build an internal model of the ever-changing world, predict its future and adapt to it. If we could understand the operation of the neocortex we might, one day, be able to construct intelligent machines capable of solving problems currently only human beings are capable of solving.

However, the neocortex is one of the most complex systems we know of – in a human consisting of around 20×10^9 neurons connected by 150×10^{12} synaptic connections [1]. In chapter 2 I give a brief overview of our current understanding of the function of the neocortex. However, furthering this knowledge through experiments on biological tissue alone is extremely difficult. Non-intrusive techniques such as functional magnetic resonance imaging (fMRI) provide only very limited temporal and spatial resolution and, while the number of neurons which can be simultaneously recorded using implanted electrodes is growing [2], it is currently limited to only a few hundred.

An alternative research direction is to build models of the brain and experiment on simulations of these models instead. Simulations not only allow us to fully control the input to the model being simulated but also to fully observe its

outputs – both impossible within the brain of a living animal. The electrophysiological behaviour of individual neurons is now relatively well understood and several large scale projects are underway [3, 4] to build "connectomes" which can be thought of as wiring diagrams of the brain. While the task of simulating models at this scale is formidable, the development of petaFLOP supercomputers means that it is becoming increasingly computationally tractable [5], albeit at the cost of megawatts. In chapter 3 I discuss the software that enables large cortical networks to be simulated on such machines and how alternative technologies such as graphics processing units (GPUs), field-programmable gate arrays (FPGAs) and neuromorphic hardware may provide less power-hungry alternatives.

SpiNNaker is a digital neuromorphic architecture designed specifically for simulating networks of up to 1×10^9 neurons in real time – enough to simulate the brains of ten mice [6]. While the fundamental computational element of a SpiNNaker system is a general purpose ARM processor, SpiNNaker systems – inspired by the brain – can combine up to a million such densely connected processors each responsible for simulating up to 1000 neurons.

Whereas, in older brain areas neurons are hard-wired to perform specific tasks, the neocortex has a relatively homogeneous structure within which neurons are densely connected by highly plastic synapses. The efficacy and structure of these synapses constantly changes in response to stimuli, allowing the neocortex to perform a wide range of functions. However due to the sheer number of these synapses and their plasticity, simulating them is a particularly complex problem which previous SpiNNaker solutions have been incapable of addressing. I have developed a new SpiNNaker synaptic plasticity implementation which has lower algorithmic complexity than prior approaches and employs new low-level optimisations to better exploit the ARM instruction set. In chapter 4 I discuss this implementation in depth and show that, not only does it almost double the perfor-

mance of previous solutions, but it is also flexible enough to support a wide range of different plasticity rules.

Bayesian inference provides an intuitive model of how our brains internalise uncertainty about the outside world and, using the Bayesian Confidence Propagation Neural Network (BCPNN) plasticity rule [7], it can be approximated using spiking neurons. In chapter 5 I present the first SpiNNaker implementation of BCPNN. Using this and a simple model of the neocortex, I demonstrate one way in which the neocortex might be able to learn and replay sequences of neuronal activity. Lashley [8] suggested that mammals' behaviour is based on hierarchical sequences of actions and experimental evidence [9] has suggested that these sequences of actions are realised as sequences of neuronal activity. Therefore the ability of my neocortical model to learn such sequences is an important step towards building functional models of the neocortex. I simulate the neocortical model at scales of up to 20×10^3 neurons and 51.2×10^6 plastic synapses: the largest plastic neural network ever to be simulated on neuromorphic hardware. However, while running this network on SpiNNaker uses less than 3 % of the power required to do so using a Cray XC-30 supercomputer system, it reveals some issues with simulating large-scale neocortical models on SpiNNaker.

In chapter 6 I analyse these issues in more depth and find that, firstly, the performance of the current SpiNNaker neural simulator scales poorly when simulating neurons with the degree of connectivity seen in the neocortex. Secondly, I find that it would be difficult to extend the synaptic plasticity implementation developed in chapter 4 to support neurons whose input synapses have different synaptic plasticity rules. I solve both of these issues by developing a novel "synapse-centric" approach for mapping large-scale spiking neural networks to SpiNNaker where the simulation of neurons and synapses is distributed amongst different cores of a SpiNNaker chip. In a benchmark network where neurons receive in-

put from a biologically-plausible number of plastic synapses, this new approach quadruples the number of neurons that can be simulated on each SpiNNaker core. I also demonstrate that, using this new approach, the neocortical model developed in chapter 5 can be run in real time: double the speed that was previously achieved.

1.1 Publications

Much of the work discussed in this thesis has previously been presented in the following publications:

J. C. Knight, P. J. Tully, B. A. Kaplan, A. Lansner, and S. B. Furber, "Large-scale simulations of plastic neural networks on neuromorphic hardware.," ***Frontiers in Neuroanatomy*****, vol. 10, no. April, p. 37, 2016, ISSN: 1662-5129** In this paper I first presented the SpiNNaker implementation of the BCPNN learning rule, the neocortical model of sequence learning and the comparisons with the supercomputer simulations discussed in chapter 5. This paper has been selected by the Faculty of Science and Engineering at the University of Manchester to be showcased as a "world leading paper".

J. Knight and S. Furber, "Synapse-centric mapping of cortical models to the SpiNNaker neuromorphic architecture," ***Frontiers in Neuroscience*****, vol. 10, p. 420, 2016, ISSN: 1662-453X** In this paper I first presented the synapse-centric simulation approach discussed in chapter 6.

A. Mundy, J. Knight, T. C. Stewart, and S. Furber, "An efficient SpiNNaker implementation of the Neural Engineering Framework," in ***The 2015 International Joint Conference on Neural Networks (IJCNN)*****, IEEE, 2015** I made modest contributions to this paper, primarily using the profiling tools devel-

oped for this thesis to analyse the performance of a novel SpiNNaker implementation of the Neural Engineering Framework (NEF) [13] developed by Andrew Mundy. This paper was nominated for the best paper award at the International Joint Conference on Neural Networks (IJCNN) 2015.

J. Knight, A. R. Voelker, A. Mundy, C. Eliasmith, and S. Furber, "Efficient SpiNNaker simulation of a heteroassociative memory using the neural engineering framework," in *The 2016 International Joint Conference on Neural Networks (IJCNN)*, 2016 This paper builds on the previous paper by investigating how two forms of synaptic plasticity [15, 16] can be incorporated into NEF models and used as the basis for large-scale heteroassociative memories which we demonstrated could be efficiently simulated on SpiNNaker. I presented this paper at IJCNN 2016 in Vancouver.

1.2 Contributions

This thesis presents the following main contributions:

Plasticity implementation I develop a new synaptic plasticity implementation for SpiNNaker which is not only almost twice as fast as previous solutions, but is also a key component of the software developed for the Human Brain Project [17].

Bayesian Confidence Propagation Neural Networks BCPNN provides a means of approximating Bayesian inference using networks of spiking neurons. Not only do I implement the learning rule required to simulate such networks on SpiNNaker, but I also use it as the basis of the largest plastic neural network ever to be simulated on neuromorphic hardware.

Synapse-centric simulation I develop this entirely new means of simulating neural networks on SpiNNaker and demonstrate that it quadruples the number of neurons with biologically-plausible numbers of plastic synapses that can be simulated on a given SpiNNaker machine.

Chapter 2

Neural systems

To model the neocortex we need to understand the fundamental biological behaviour of both the neurons within it and of the circuits they form. In section 2.1 I present a brief background on how biological neurons communicate and process information and in section 2.2 I present evidence on the structure of the neocortex and theories on how this may enable it to perform high-level cognitive functions.

2.1 Neurons

Like all animal cells, neurons consist of an impermeable cell membrane surrounding a nucleus, mitochondria and other organelles. However, neurons also have several specialised components which support electro-chemical computation and signalling. *Ion pumps* "pump" ions across the cell membrane at an approximately constant rate to maintain a "resting" potential difference between the cell's interior and the extracellular fluid which surrounds it. *Ion channels* also enable ions to cross the cell membrane but, unlike ion pumps, the rate at which ion channels transfer ions can be modulated by factors including the membrane potential and the concentration of various chemical messengers.

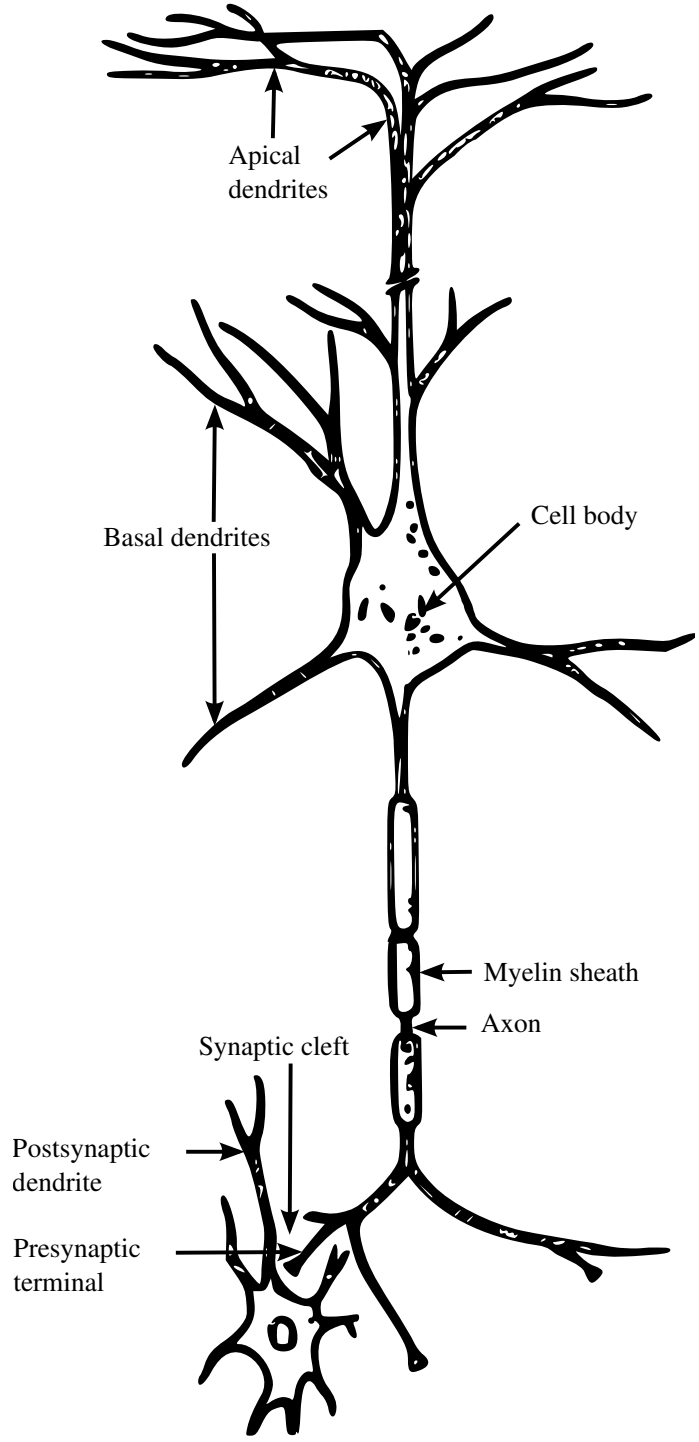


Figure 2.1: Structure of a pyramidal neuron. After [18].

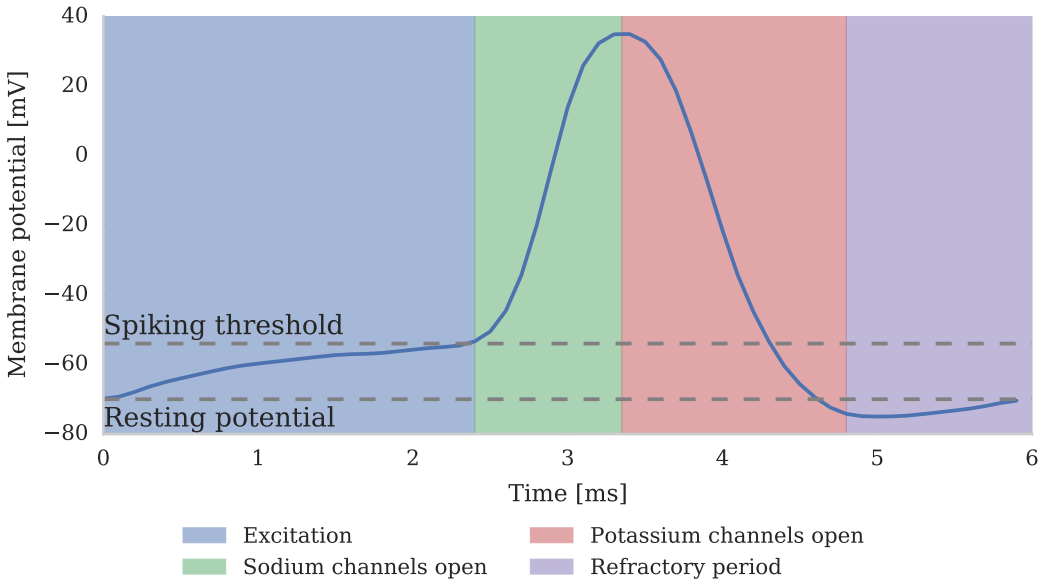


Figure 2.2: A spike. Dashed horizontal lines show the resting potential and the threshold at which sodium ion channels open.

Figure 2.1 shows the structure of a *pyramidal neuron* – one of the most common neuron types found in the neocortex. Changes in a neuron's membrane potential cause sodium and potassium ion channels to act together, generating *spikes*: the primary means of neuronal communication. When the membrane potential reaches a certain threshold, sodium ion channels open allowing positively charged sodium ions to rush into the neuron, rapidly increasing the membrane potential from around −70 mV to around 30 mV. This sudden increase in membrane potential causes potassium ion channels to open, allowing positively charged potassium ions to leave the neuron, and resulting in the stereotypical spike in membrane potential potential shown in figure 2.2. Spikes then propagate down the *axon* of the neuron as a wave of sodium and potassium ion channels open ahead of the spike. As the pyramidal neuron in figure 2.1 illustrates, axons can also be wrapped in a myelin sheath, through which spikes can propagate at much higher speeds. Be-

fore the neuron can spike again, ion pumps need to pump sodium ions out of the neuron and potassium ions back in, restoring the membrane potential to its resting level – a period known as the *refractory period*. To communicate spikes to other neurons connections known as *synapses* are formed at the *axon terminals*.

From the point-of-view of one of these synapses, the neuron from which spikes originate is known as the *presynaptic* neuron and the neuron which has made the synaptic connection is known as the *postsynaptic* neuron. In general spikes can have two effects on a postsynaptic neuron: *excitatory* spikes increase its membrane voltage and *inhibitory* spikes decrease it. Dale's law [19] states that a neuron cannot excite some of its postsynaptic targets and inhibit others. Therefore excitatory spikes are always emitted by excitatory neurons (such as the pyramidal neuron shown in figure 2.1) and inhibitory spikes are always emitted by inhibitory neurons. When a spike reaches an axon terminal, the increase in membrane potential causes calcium ion channels to open, drawing calcium ions into the neuron. The resultant increased calcium concentration inside the neuron causes neurotransmitters to be released into the *synaptic cleft* between the pre and postsynaptic neurons: primarily glutamate from excitatory neurons and GABA from inhibitory neurons. These neurotransmitters diffuse across the synaptic cleft and bind to receptors, causing a final class of ion channels to open on the postsynaptic neuron. The ion channels that open as result of glutamate binding allow current to flow into the neuron and those that open as a result of GABA binding allow current to flow out. Two important classes of ion channels that open in response to glutamate binding are known as AMPA and NMDA. AMPA ion channels open and close relatively quickly whereas NMDA ion channels not only open and close slower, but also require the neuron's membrane voltage to be elevated prior to activating. In section 5.5 I present a model which uses the different time constants of AMPA and NMDA synapses to learn features of temporal sequences at different time scales.

Synaptic connections can be made to almost any part of the neuron and how the position of these connections affects the integration of the input currents they supply is not fully understood. However, using the geometry of the neuron shown in figure 2.1 as an example, synaptic input applied directly to the cell body or to the nearby *basal dendrites* is likely to have more effect on spike generation than synaptic input applied to the *apical dendrites* further from the cell body [20]. There is also evidence that local computation occurs within dendritic branches [21] and that the interactions between NMDA ion channels and AMPA ion channels have an important role [22].

2.2 The Neocortex

The neocortex is the most recent part of brain to evolve and consists of a sheet of neurons ranging in size from around 6 cm^2 in a rat to 2500 cm^2 in a human. While in older brain regions, specialised neurons and synaptic connections are "hard-wired" to perform specific functions, the neocortex has a relatively homogeneous structure with highly plastic synaptic connections whose efficacy and structure constantly change in response to stimuli. This perhaps reflects the fact that, rather than performing a single function, much of the processing performed by the neocortex is learnt postnatally [23]. In fact, even in adulthood, the neocortex remains plastic enough to allow vision to be restored – after sufficient training – by projecting images from a camera onto a matrix of actuators attached to the skin [24].

The surface of neocortex is divided into areas responsible for processing visual, auditory and somatosensory stimuli; generating motor commands and performing higher-level cognitive functions. Within each of these areas there is then a hierarchy of *regions* connected together with both *feedforward* and *feedback* connections. Typically this hierarchy reflects an increasing level of information

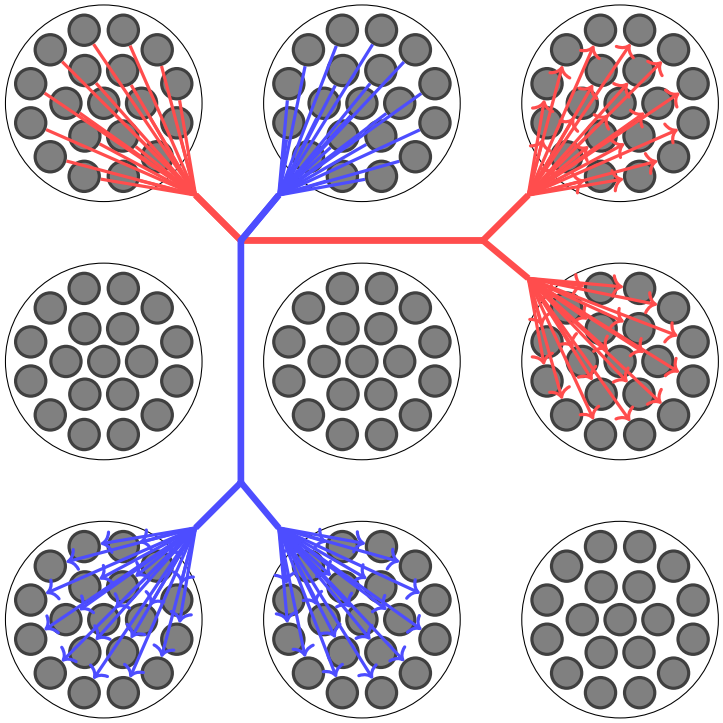


Figure 2.3: Neocortical columnar connectivity. Minicolumns (filled grey circles) arranged into 9 macrocolumns (large outlined circles) and connected with patchy connectivity.

abstraction within each region. For example the primary visual cortex – the lowest level of the visual cortex hierarchy – receives input relayed via the thalamus from the retina and responds to oriented line segments crossing the visual field [25] whereas the inferior temporal cortex – at the top level of the visual cortex hierarchy – has been shown to respond to high-level concepts such as human faces [26].

Each neocortical area is made up of a large number of *macrocolumns* with diameter in the range of 300 µm to 600 µm. Each of these macrocolumns consists of approximately 100 *minicolumns* bound together with short-range horizontal connections. Minicolumns are believed to be the basic functional unit of the neocortex with each one consisting of approximately 100 neurons, tuned to respond to the same stimuli [27, 28].

Across species and neocortical areas the size and structure of macrocolumns remain approximately constant – the neocortices of humans and higher primates simply have a larger number of macrocolumns than, for example, those of mice. The average number of synapses providing input to each cortical neuron also remains constant across species at around 8000 [1, 29, 30]. These properties are independent of the size of the neocortex because cortical long-range cortical connectivity follows the "patchy" structure illustrated in figure 2.3 [31–34]. The "fan-out" of each minicolumn varies between neocortical areas but, for example in the primary visual cortex, each minicolumn only connects to around 10 macrocolumns located within a radius of a few millimetres. The density of connections between individual neurons in the connected minicolumns varies widely but is always relatively sparse (recent measurements in the somatosensory cortex of rats [35] suggest that pyramidal neuron connectivity saturates at around 20 %).

Hubel and Wiesel [25] provided the first experimental evidence of the role of macrocolumns in the primary visual cortex of Macaque monkeys. These macrocolumns apply what has become known as a winner-take-all (WTA) circuit to the output of the minicolumns which respond to the orientation of line segments. The WTA circuit inhibits the output of all of minicolumns in a macrocolumn aside from the one responding most strongly to the stimuli. While in higher cortical areas input cannot be directly manipulated, making it extremely difficult to perform equivalent experiments, Douglas and Martin [36] suggest that this WTA circuitry is fundamental to all neocortical areas.

Perpendicularly to this columnar structure the neocortex is also divided into 6 layers which, as illustrated in figure 2.4, can be grouped as follows:

Layer I contains very few neurons and its role appears to be largely one of "scaffolding" – a location where feedback synapses from higher cortical areas can make connections with the dendrites of neurons in layers II/III.

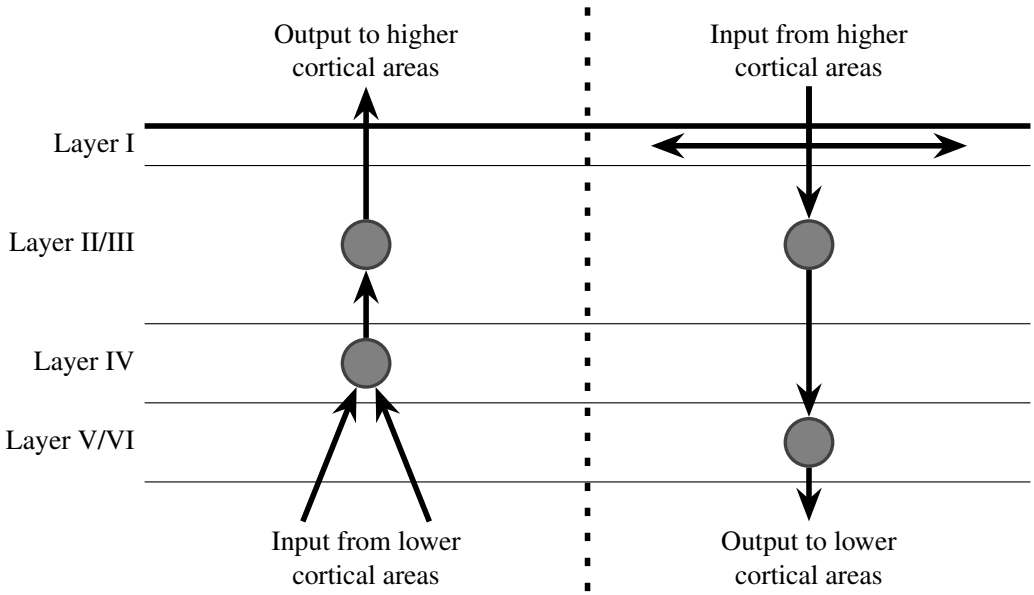


Figure 2.4: **Left**: upward and **Right**: downward flow of information between layers of the neocortex. Filled grey circles represent neurons taking part in the information flow.

Layer II/III are where associations between feedback and feedforward input are made; and fed forward to higher cortical areas and back via layers V/VI.

Layer IV is where feedforward input enters the macrocolumn from lower cortical areas and is passed to layers II/III.

Layer V/VI passes the feedback output from layers II/III on to the dendrites located in layer I of lower cortical areas.

However, even if we accept this cortical architecture, exactly how the brain uses this structure to implement high-level cognitive functions remains an open question.

Chapter 3

Modelling Neural Systems

Section 2.1 presented a high-level overview of the electrochemical behaviour of individual neurons and in section 3.1 I will discuss how this behaviour can be modelled at various levels of abstraction. In section 2.2 I went on to discuss how neurons in the neocortex are arranged in a highly-connected, three dimensional structure. This high degree of connectivity makes simulating large cortical networks particularly challenging and in sections 3.2 - 3.5 I will give an overview of some of the approaches that have been applied to this problem.

SpiNNaker is a digital neuromorphic architecture designed specifically for the simulation of large-scale neural networks and in section 3.7 I will discuss the SpiNNaker system and its performance characteristics in more depth.

3.1 Neural modelling

As figure 2.1 illustrates, biological neurons can have complex dendritic tree structures. However the classical view on their role, as described by Ramón y Cajal [37], is that:

> "Dendrites exist solely to allow the cell to receive, and then transmit to the axon, the greatest variety of information from the most diverse sources."

From this assumption the entire neuron can be viewed as being somewhat analogous to a logic gate: gathering information from multiple sources and generating a single output in the soma (see figure 2.1). Models based on this assumption are known as *point neuron models* and in section 3.1.1 I present a brief overview of models of this type. In the remainder of this thesis I concentrate largely on simulating point neuron models. However it has been suggested that dendritic structure may play an important role in neurons' information processing functionality. Therefore in section 3.1.2 I briefly discuss how point neuron models can be extended to form *multi-compartmental models* that can be used to model neurons with more realistic dendritic structures.

3.1.1 Point neuron models

Binary models

Coming from the logic gate analogy it is unsurprising that early computational models of neurons, such as the Threshold Logic Unit (TLU) developed by McCulloch and Pitts [38], closely resemble logic gates. The TLU receives multiple binary inputs (x_i) and produces a single binary output (y) by applying a Heaviside step *activation function* to the sum of a bias term (β) and the inputs (x_i), weighted with the weights (w_{ij}):

$$u = \beta + \sum_{i=1}^{n} w_i x_i \tag{3.1}$$

$$y = \begin{cases} 1 & if\ u \geq 0 \\ 0 & if\ u < 0 \end{cases} \tag{3.2}$$

This neuron model can be directly used to implement boolean AND and OR functions and, by building recurrent networks, where the output of some neurons fed back into others, it could be used to build simple memory systems.

Rate-based neuron models

The TLU model was subsequently generalised to use real valued inputs. In this case, the w_i and β parameters can be thought of as defining a plane through an n-dimensional input space, which defines a linear separation between two classes of input vectors. Neurons of this type have been used to build single layer artificial neural networks (ANNs) such as Perceptrons [39] which can be combined to form multi-layer networks capable of classifying patterns that are not linearly separable. To give these extra layers more computational power than is achievable by using a larger layer of linear neurons multi-layer networks typically use neurons that generate a real valued output based on a non-linear activation function. Furthermore to enable the use of the back-propagation algorithm [40] – a popular approach for training multi-layer ANNs – these activation functions are required to be differentiable. Therefore activation functions such as tanh or the sigmoid function [41] are common choices. In terms of modelling biological neurons this output can be thought of as representing the firing rate of the neuron.

Spiking neuron models

Although rate-based neuron models are mathematically convenient and the firing rates of neurons have been shown to encode stimuli [42] in many parts of the brain, both temporal [43] and spatial information [44] has also been shown to be encoded in the timings of individual spikes. To model these effects, more realistic spiking models must be used.

As discussed in section 2.1, neurons emit spikes when their membrane potential is rapidly depolarised by the opening of sodium ion channels before being immediately repolarised by the opening of potassium ion channels. The potential across an ion channel can be modelled using the following simple ODE:

$$C\frac{dV}{dt} = -g_{ion}\left(V - E_{ion}\right) \tag{3.3}$$

Where C is the membrane capacitance, V is the potential across the ion channel, G_{ion} is the conductance of the ion channel and E_{ion} is the channel's reversal potential. Hodgkin and Huxley [45] developed a neural model which combined the effect of 3 types of ion channel: a leak channel (L) with a constant conductance representing the action of the ion pumps and voltage-gated potassium (K) and sodium channels (Na):

$$C\frac{dV}{dt} = -g_K n\left(V - E_K\right) - g_{Na} m^3 h\left(V - E_{Na}\right) - g_L\left(V - E_L\right) \tag{3.4}$$

The voltage gating is achieved using the dimensionless n, m and h variables, each of which is modelled using a further differential equation:

$$\frac{dn}{dt} = \frac{n_\infty(V) - n}{\tau_n(V)} \qquad \frac{dm}{dt} = \frac{m_\infty(V) - m}{\tau_m(V)} \qquad \frac{dh}{dt} = \frac{h_\infty(V) - h}{\tau_h(V)} \tag{3.5}$$

Where n_∞, m_∞, h_∞, τ_n, τ_m and τ_h represent different, voltage-dependent functions.

Hodgkin-Huxley neurons, fitted to biological data, have been used for large-scale modelling [5], but are extremely computationally expensive to simulate and have a large number of parameters to configure. Therefore a common alternative approach is to model the subthreshold dynamics of the membrane voltage – up to the point at which the neuron emits a spike – using a simpler system and then artificially reset the membrane voltage to simulate rapid repolarisation. One of the simplest models of this type is the leaky integrate-and-fire (LIF) model whose subthreshold membrane voltage (V) is modelled using the following simple first-order ODE:

$$C\frac{dV}{dt} = -g_L(V - V_{rest}) + I_{app} \tag{3.6}$$

Where g_L and C represent the leak conductance of the membrane and its capacitance; V_{rest} represents the membrane's resting voltage and I_{app} represents the input current. In the absence of any input, V decays linearly with a time constant of $\tau_m = \frac{C}{g_L}$. However when V reaches a fixed threshold (V_{thresh}) it is reset to V_{reset} and a spike is emitted. Additionally a timer can be started to prevent further input being integrated during the period in which a biological neuron would be in its refractory period. The LIF model has the advantage of being fast to compute and, as equation 3.6 has an algebraic solution, easy to analyse mathematically.

However, as figure 3.1 shows, the LIF model will always spike regularly when provided with a constant input current whereas cortical neurons have been shown

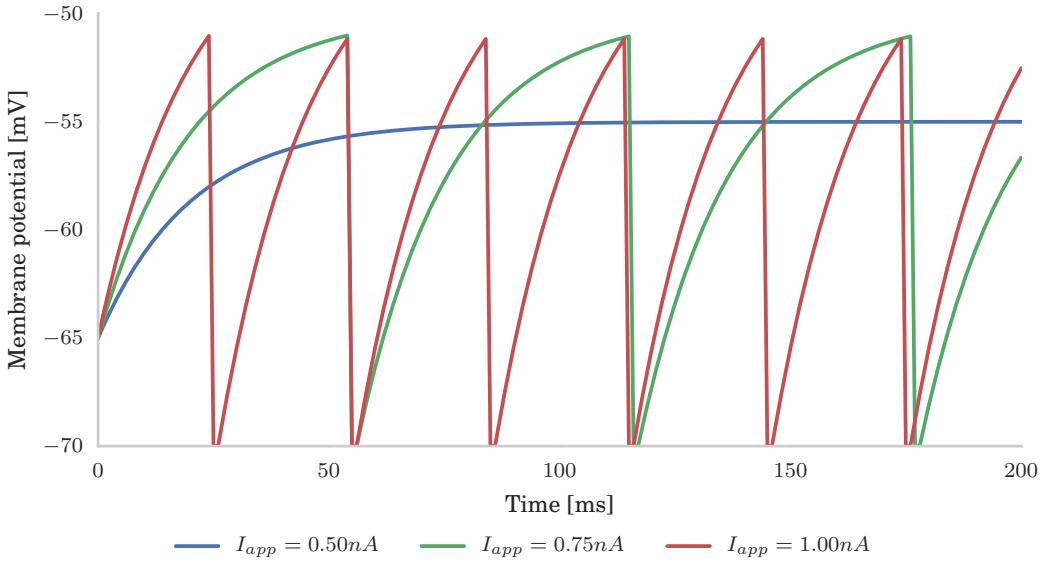


Figure 3.1: Response of a LIF neuron to a constant input current. Simulated using NEST 2.6 [46] with $V_{thresh} = -51\,\text{mV}$, $V_{reset} = V_{rest} = -70\,\text{mV}$, $C = 1\,\text{nF}$, $g_L = 50\,\text{nS}$.

to reproduce a wide range of spiking behaviours [47–49].

By modelling the subthreshold neural dynamics using a more complex two dimensional system Izhikevich [50] developed an integrate-and-fire neuron model which can replicate a much wider range of spiking behaviours while remaining fast to compute:

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I_{app} \tag{3.7}$$

$$\frac{du}{dt} = a\,(bV - u) \tag{3.8}$$

Similarly to the LIF model, when V reaches a threshold, it is reset to a fixed value and u has an offset applied to it. The quadratic term of equation 3.7 enables the membrane voltage to rise in a faster, more realistic manner than that of the

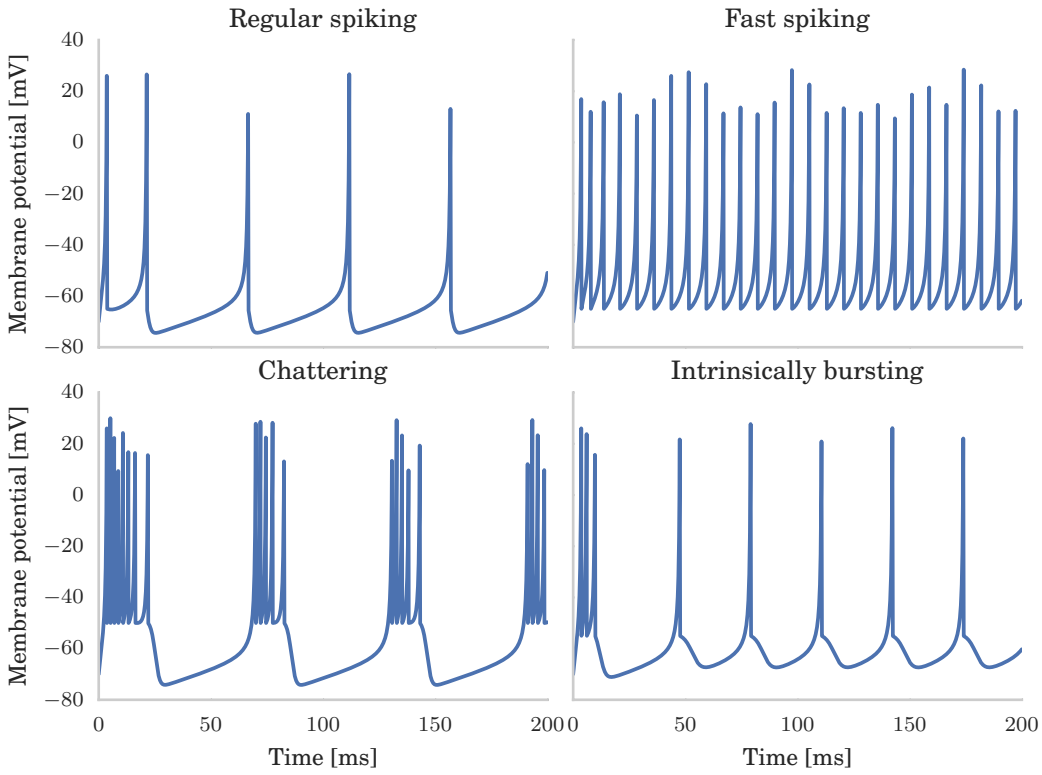


Figure 3.2: Response of a Izhikevich neuron to a constant 10 pA input current. Simulated using NEST 2.6 [46] using parameters specified by Izhikevich [50].

LIF neuron during the period immediately before a spike. Additionally, the u state variable acts as a subthreshold adaptation mechanism – altering the firing rate of the neuron based on its past activity [51]. By using different values of the a, b, c and d parameters the Izhikevich neuron can reproduce the wide range of spiking behaviours illustrated in figure 3.2.

The adaptive exponential integrate-and-fire model (AdEx) [52] is another widely used neuron model and has been shown to accurately reproduce the behaviour of neurons modelled using the Hodgkin-Huxley equations. As with the Izhikevich model, the aEIF model is described by two coupled differential equations:

$$C\frac{dV}{dt} = -g_L(V - V_{rest}) + g_L\Delta_T \exp\left(\frac{V - V_{thresh}}{\Delta_T}\right) - w + I_{app} \tag{3.9}$$

$$\tau_w\frac{dw}{dt} = a(V - V_{rest}) - w \tag{3.10}$$

As with the LIF model, g_L and C represent the leak conductance of the membrane and its capacitance; V_{rest} represents the membrane's resting voltage; V_{thresh} the spiking threshold and; I_{app} the input current. The second w state variable plays the same role as u in the Izhikevich model – representing a subthreshold adaptation mechanism with a strength defined by a and a time constant of τ_w. Finally Δ_T defines the "sharpness" of the spiking threshold. The behaviour of the aEIF model at this threshold is somewhat different from that of the LIF and Izhikevich models as, when $V \geq V_{thresh}$, the exponential term causes the membrane to rapidly depolarise before it is artificially reset at a second, higher threshold $V = 20\,\text{mV}$. At this point, similarly to the Izhikevich model, V is reset to V_{reset} and b is added to w (defining the strength of the spike-triggered adaptation).

In all of the integrate-and-fire neuron models discussed in this section I have represented the input to the neuron model as a single I_{app} term. While a constant

value of I_{app} can be used to perform the kind of simple experiments shown in figures 3.1 and 3.2, typically I_{app} is calculated from the conductances of the neuron's synapses using equation 3.3. Although synaptic conductance injection has been shown to better represent the behaviour of biological neurons [53], to make the model easier to analyse mathematically this additional non-linearity can be removed and synapses can be viewed as injecting current directly.

3.1.2 Multi-compartmental neural models

The point neuron models described in the previous section all assume that the membrane potential of a neuron is constant across its entire cell membrane. However, particularly when neurons have narrow sections such as those within the dendritic tree of the pyramidal neuron shown in figure 2.1, this may be a poor approximation. *Multi-compartmental models* address this by splitting the neuron into smaller *compartments* within which the membrane voltage is assumed to be constant. Each of these compartments can then be simulated using either a detailed Hodgkin-Huxley or a simpler integrate-and-fire model [54]. Compartments are then linked using ohmic channels allowing current to flow between neighbouring compartments [55, p. 217].

3.1.3 Synaptic input

As discussed in section 2.1, spike transmission between biological neurons is the result of a complex cascade of biological processes. Similarly to the Hodgkin-Huxley models of neurons discussed in section 3.1.1 highly detailed models of synaptic transmission have been developed [56] but, as with detailed neuron models, are costly to simulate. Therefore simpler models of the conductance of the postsynaptic ion channel are often employed.

The simplest models of ion channel conductance assume that a presynaptic spike at time t_0 causes the ion channel to open immediately and its synaptic conductance (g_{syn}) to increase immediately to $\bar{g}_{syn}$ before exponentially decaying with time constant τ as the neurotransmitters are reabsorbed and the ion channel closes:

$$g_{syn}(t) = \bar{g}_{syn} \exp\left(-\frac{t - t_0}{\tau}\right) \tag{3.11}$$

However, as the rise time of a real ion channel's conductance is finite, it is more realistic to describe this conductance using the difference of two exponentials:

$$g_{syn}(t) = \bar{g}_{syn} c \left(\exp\left(-\frac{t - t_0}{\tau_{decay}}\right) - \exp\left(-\frac{t - t_0}{\tau_{rise}}\right)\right) \tag{3.12}$$

Where c is a constant used to scale the peak amplitude to $\bar{g}_{syn}$; and τ_{rise} and τ_{decay} are the time constants of the ion channel opening and closing respectively. In the special case where the peak synaptic conductance occurs at $t = \tau$ this model can be simplified to an *alpha* function with a single time constant τ:

$$g_{syn}(t) = \bar{g}_{syn} \frac{t - t_0}{\tau} \exp\left(1 - \frac{t - t_0}{\tau}\right) \tag{3.13}$$

3.2 Software simulation

In this section I will outline some strategies used to simulate spiking neural networks in software before briefly discussing a representative selection of the tools which use them.

Clock-driven simulation algorithms, where time is advanced in discrete steps (typically 1 ms or 0.1 ms), are amongst the most common. In a typical clock-driven simulation algorithm the state of each neuron is updated every time step using some form of numerical integration technique such as Runge-Kutta [57]. However some models, such as simple forms of the LIF neuron discussed in section 3.1.1, are linear so can be solved by simply multiplying the previous state by a constant matrix [58]. After updating each neuron in this manner its spike threshold condition is checked and, if it is satisfied, the spike is propagated to all target neurons. While this spike propagation technique is convenient because "time models itself", its results are only approximate as spike threshold conditions are only tested at time step boundaries, and therefore exact spike timings are lost. The clock-driven simulation scheme can be extended to overcome this inaccuracy by calculating the times at which neurons spike within the simulation time step and passing these times, along with the spikes, to the target neurons. Neurons can then sort incoming spikes based on these times and apply them exactly.

An alternative approach is to employ an event-driven algorithm where the state of neurons and synapses are updated only when they receive spikes. This alleviates the inaccuracies caused by clock-driven approaches and has the potential to improve performance if spikes are sufficiently sparse. However, event-driven algorithms require that neuron and synapse models have explicit solutions i.e. that their state can be calculated at any time. All the static synapse models discussed in section 3.1.3 and many plastic synapse models (discussed further in chapter 4) have this property. However, of the neuron models discussed in section 3.1.1, only the LIF model has an explicit solution [59, 60]. Furthermore Morrison *et al.* [61] argue that, because cortical neurons receive input from the order of 10×10^3 other neurons, the combined input event rate is such that any computational advantages of using an event-driven algorithm for simulating neurons is lost. Therefore many

software simulators – including NEST [46], NEURON [62] and Brian [63] – use a "hybrid" approach where neurons are simulated using a time-driven algorithm and synapses using an event-driven algorithm.

One common means of accelerating large-scale software simulations is by running them on distributed cluster systems or even supercomputers. Both NEST and NEURON support this mode of operation and have both been shown to enable supra-linear speedup across thousands of processors [64]. In fact NEST was used to run the largest neuronal network simulation to date consisting of 1.73×10^{12} neurons and 10.4×10^{18} synapses distributed across 82 944 processors of the K supercomputer and taking 40 min to simulate 1 s of neural activity [65].

3.3 GPU simulation

With 69 of the machines in the June 2016 Top 500 list [66] featuring GPU acceleration it is clear that GPUs have become a dominant force in the high-performance computing (HPC) landscape. Therefore, as a major HPC application, it is unsurprising that there has been significant interest in using GPUs to accelerate spiking neural network simulations.

Nageswaran *et al.* [67] developed a GPU simulator which was used to simulate a network with up to 100×10^3 Izhikevich neurons and 10×10^6 synapses at 0.66× real time. Fidjeland and Shanahan [68] further optimised this approach to enable neurons with more realistic numbers of synapses to be simulated and used this to simulate a network with 30×10^3 Izhikevich neurons and 30×10^6 at approximately 2× real time. However neither of these approaches offers a large speedup over a CPU implementation considering the raw computational power GPUs offer (Nageswaran *et al.* reported 26×).

Both Nageswaran *et al.* and Fidjeland and Shanahan identify that this bottle-

neck arises largely because synaptic transmission makes inefficient use of GPU architectures. Yavuz *et al.* [69] provided further evidence for this inefficiency by simulating networks of both Izhikevich and Hodgkin-Huxley neurons. They found that their GPU implementation offered a speedup of over 100× in simulations using Hodgkin-Huxley neurons where updating each neuron is a mathematically intensive operation. However they obtained only a 10× speedup in simulations of a simple cortical network built with integrate-and-fire neurons where synaptic transmission costs dominated. The neocortical models which are the focus of this thesis have properties much more similar to this second network and, as GPUs can also have a peak power usage of around 200 W, they seem ill-suited to the low power simulation of such models.

3.4 Neuromorphic hardware

As discussed in section 3.2, software simulators can scale to take advantage of the latest peta-scale supercomputers, allowing extremely large neural networks to be simulated. However supercomputers, like the GPUs discussed in the previous section, require considerable electrical power to do this.

In the late 1980s Mead [70] observed that transistors exhibit "hauntingly" similar electrical behaviour to neurons and synapses. Therefore Mead proposed that, instead of using power-hungry digital computers to simulate the brain, we could instead implement neurons and synapses using sub-threshold analogue circuits. This approach became known as "neuromorphic engineering" and, over the proceeding years, a number of neuromorphic systems have been built with the aim of reducing the power consumption and execution time of neural simulations. These systems have been constructed using a number of approaches: ROLLS [71], NeuroGrid [72] and BrainScaleS [73] are built using custom analogue hardware and

True North [74] is built using custom digital hardware. These systems all have the potential to simulate neural networks using many orders of magnitude less power than the software-based approaches discussed in the previous section. However, they also all share several limitations which hinder their ability to simulate the type of highly-connected, plastic models of the cortex that are the focus of this thesis. These limitations stem from how synapses are implemented in such systems. Both BrainScaleS and TrueNorth are fabricated with several hundred individual synapse circuits associated with each neuron. Although both systems have mechanisms that allow synapses to be "borrowed" from other neurons to implement higher connectivity, this is done at the expense of the total number of neurons the system can simulate. ROLLS and NeuroGrid use an alternative approach where each neuron has a single synaptic input circuit into which input currents from multiple synapses are injected. While this approach allows much higher degrees of connectivity to be achieved, synapses implemented in this manner cannot support plasticity. To overcome this limitation ROLLS also has a limited number of plastic synapses implemented as individual circuits.

3.5 FPGA simulation

The development of application specific integrated circuits (ASICS) of the type discussed in section 3.4 has become prohibitively expensive as process sizes shrink [75]. Therefore field-programmable gate arrays (FPGAs) – devices consisting of a large number of lookup-table based logic blocks, connected using a programmable fabric – are becoming a popular alternative in many small-volume applications. However the additional logic required to implement this programmability means that a given circuit may occupy 40× more silicon area if it is implemented using FPGA rather than ASIC logic [76]. Therefore employing the approach used

by the neuromorphic devices discussed in section 3.4 where individual neuron circuits are implementing in FPGA logic only scales to very small networks of neurons [77].

However modern FPGAs typically run at clock speeds in the order of hundreds of MHz: many times faster than biological neurons. Therefore, as simple integrate-and-fire neurons of the type discussed in section 3.1.1 can be evaluated in very few clock cycles, a single neuron circuit can be used to simulate large number of "time-division multiplexed" neurons. Moore *et al.* [78] used this approach in the multi-FPGA Bluehive system which is capable of simulating up to 64×10^3 Izhikevich neurons and 64×10^6 synapses on a single FPGA. Furthermore Moore *et al.* predict that the performance of Bluehive will scale linearly across up to 64 FPGAs. However the resultant system is still somewhat inflexible as the neuron model is directly implemented in FPGA logic.

One way in which FPGA designs can be made more flexible is by incorporating a general purpose CPU, implemented in FPGA logic, into the design so that non-performance-critical components can be implemented in software. Naylor *et al.* [79] developed a vector coprocessor which improves the performance of such a CPU to the point where it can accelerate neural simulations to within a factor of two of the Bluehive system. This approach shows much promise but it remains to be seen how competitive it would be at the scale of the SpiNNaker system discussed in section 3.7.

3.6 PyNN

In the previous sections I have discussed a variety of simulation tools with different advantages and disadvantages. However researchers wishing to work across a number of these will typically have to learn new programming languages, APIs and possibly modelling paradigms.

While it is not a simulator in its own right the PyNN [80] project aims to address this issue by providing a standard abstraction layer for describing and simulating neural network models in Python. By using this abstraction layer a model defined in PyNN can, at least in principle, be run unmodified on any supported simulator. PyNN defines models in terms of *populations* and *projections*.

Populations are logical groups of neurons which can be simulated using the same neuron model. For example, in listing 3.1 two populations are created, one contains a single LIF neuron and the other a single *SpikeSourceArray* (an artificial neuron model that injects spikes into the network at pre-programmed times).

Projections represent connections between populations which can be simulated using the same synapse model. Projections also have an associated "connector" which defines how the individual neurons within the population are connected. For example, the *AllToAllConnector* used in listing 3.1, connects each neuron in the presynaptic population to every neuron in the postsynaptic population. PyNN also provides standard interfaces for controlling the simulation and recording its state.

3.7 SpiNNaker

SpiNNaker [81] is a massively parallel computer architecture which, inspired by the biological brain, has abandoned three features common to the majority of such architectures: memory coherence, synchronicity and determinism. Instead it has

Listing 3.1: Simple example model described using PyNN 0.7. The model consists of a spike source programmed to emit a single spike 0 ms into the simulation. The spike source is connected to a single leaky integrate-and-fire neuron whose output spikes and membrane voltage are recorded.

```
import pyNN.nest as p

# Configure simulation time step to 1ms
p.setup(timestep=1.0)

# Create two populations
pop_1 = p.Population(1, p.IF_curr_exp, {})
pop_2 = p.Population(1, p.SpikeSourceArray, {"spike_times": [0]})

# Connect the two populations together
p.Projection(pop_2, pop_1, p.AllToAllConnector(weights=5.0, delays=1))

# Record neural population's spikes and membrane voltage
pop_1.record()
pop_1.record_v()

# Run simulation for 10ms
p.run(10)
```

been designed specifically for the low-power, real-time simulation of large networks of spiking neurons. Therefore, in our taxonomy of neural simulation techniques, SpiNNaker falls somewhere between the software simulators discussed in section 3.2 and the hardware simulators discussed in section 3.4.

3.7.1 Hardware

The SpiNNaker architecture can be used to build systems ranging in size from single boards to room-size machines, all using the same basic building block: the SpiNNaker chip. As shown in figure 3.3, a SpiNNaker chip contains 18, 200 MHz, ARM cores, each equipped with two small tightly-coupled memories: 32 KiB for instructions (ITCM) and 64 KiB for data (DTCM). The cores within a chip connect to each other, 128 MiB of external SDRAM and a multicast router using a network-on-chip (NoC) known as the "System NoC". Every chip's router connects

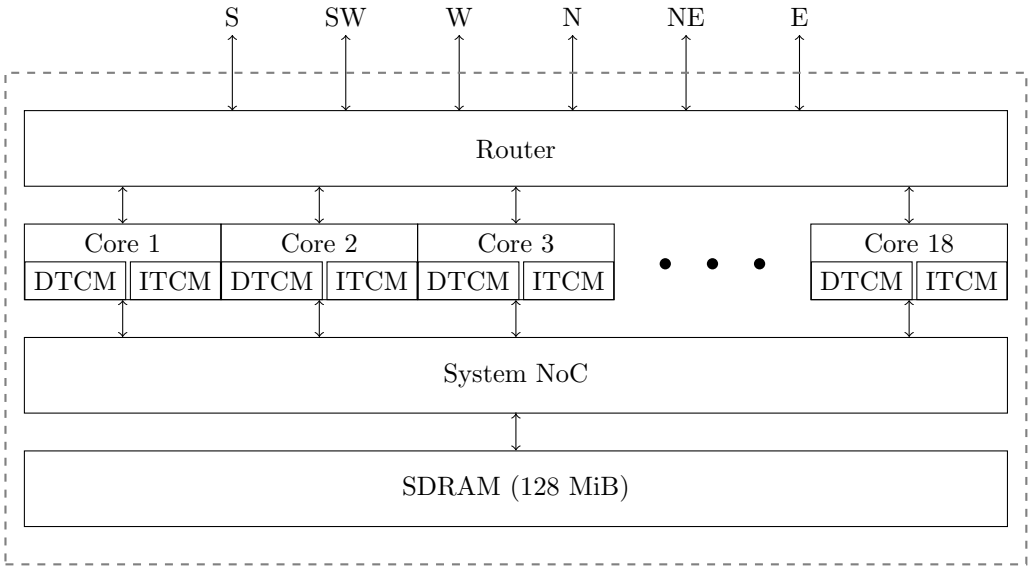


Figure 3.3: The basic architecture of a SpiNNaker chip.

to the routers of six immediate neighboring chips using a second NoC known as the "Communications NoC".

Beyond its lack of globally shared memory one notable restriction of the SpiNNaker architecture is that, for reasons of silicon area and energy efficiency, no hardware floating point unit is included. While floating point operations can be emulated in software, this incurs a significant performance penalty meaning that performance-critical SpiNNaker software needs instead to use fixed-point arithmetic. Hopkins and Furber [82] discuss the challenges of using fixed-point arithmetic for neural simulation on the SpiNNaker platform in detail and highlight two main issues of particular importance. Firstly, the range of fixed-point numeric representations is static so, to attain maximal accuracy, the optimal representation for storing each state variable must be chosen ahead of time. Secondly, there is no standard means of calculating transcendental functions such as exp or log using fixed-point arithmetic. These functions can be approximated using, for instance, a Taylor series expansion. However, the resultant functions are likely to take the

order of 100 CPU cycles to evaluate [83], making them too slow for use in the most performance-critical SpiNNaker applications. Another approach is to use pre-calculated lookup tables (LUTs). These are particularly well suited to implementing periodic functions such as $\sin(x)$ or functions such as $\exp\left(\frac{-t}{\tau}\right)$ which (for small values of τ) decay to 0 after only a small number of table entries.

3.7.2 Spiking neural network simulation

While SpiNNaker has a somewhat unusual memory hierarchy, its lack of global shared memory means that many of the problems related to simulating large spiking neural networks on a SpiNNaker system are shared with more typical distributed computer systems. Therefore the SpiNNaker neural simulator uses a very similar hybrid event and clock-driven approach to software simulators designed to run on distributed systems (section 3.2). Figure 3.4 illustrates how neural networks are mapped to SpiNNaker with each processing core being responsible for simulating a collection of neurons and their afferent synapses. The neurons are simulated using a time-driven approach with their state held in the DTCM. Each neuron is uniquely identified by a 32 bit ID and, when a simulation step results in a spike, a packet containing this ID is sent to the SpiNNaker router. These "spike" packets are then routed across the network fabric to all the cores on which neurons targeted by the spiking neuron are simulated. While, as discussed in section 3.2, this strictly clock-driven approach does not produce exact results, Hopkins and Furber [82] developed some extensions to the basic clock-driven algorithm which improve accuracy.

Due to the large number of synapses and the relatively low firing rate of single neurons, the synapses are simulated in an event-driven manner, meaning that they get updated only when they transfer a spike. On SpiNNaker this event-driven approach is also advantageous as, due to the sheer number of synapses, per-synapse

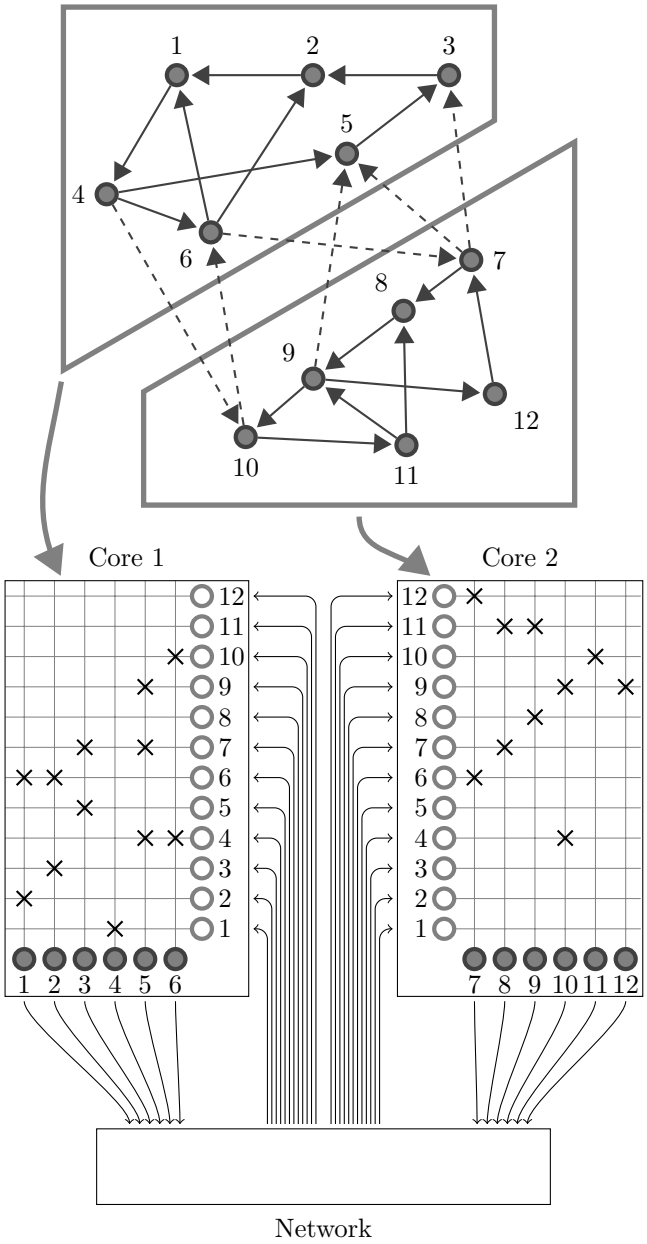


Figure 3.4: Standard mapping of a spiking neural network to SpiNNaker. An example network consisting of 12 neurons is distributed between two SpiNNaker cores. The synaptic matrix is split vertically and its columns are distributed between the two cores responsible for simulating the corresponding postsynaptic neurons (filled circles). Both cores contain synaptic matrix rows corresponding to all 12 presynaptic neurons (non-filled circles). The SpiNNaker router routes spikes from firing neurons (filled circles) to the cores responsible for simulating the neurons these spikes target.

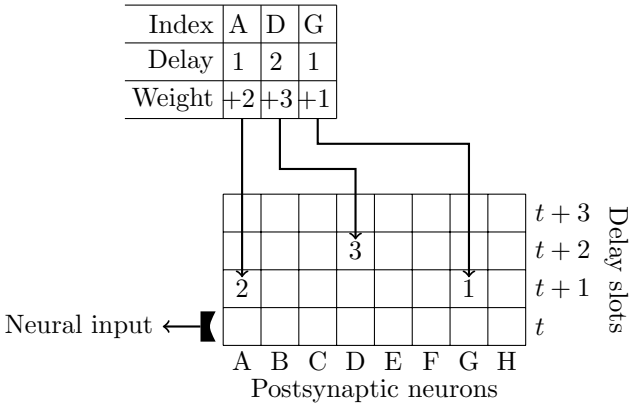


Figure 3.5: Unpacking a sparse synaptic matrix row with three synapses into a delay ring-buffer supporting delays of up to 3 simulation time steps and 8 neurons (labelled A–H).

data such as synaptic weights must be stored in the off-chip SDRAM which each core can only access at a maximum of 300 MiB s^{-1} [84] – insufficient to transfer the per-synapse data associated with every synapse each simulation time step. Instead, on receipt of a "spike" packet, cores initiate a direct memory access (DMA) transfer to fetch the row of the connectivity matrix associated with the firing neuron from SDRAM. Each of these rows describes the synaptic connections between a presynaptic neuron and the postsynaptic neurons simulated on the core. As shown in figure 3.5, rows are represented using a sparse format where each synapse consists of a *weight* (the magnitude of the input conductance or current change the spike induces), the index of the target postsynaptic neuron and a transmission delay. This transmission delay is implemented using a data structure we call a *ring-buffer* which accumulates the weights due to be applied to each neuron in future simulation time steps. Once a row is retrieved, the synaptic weights it contains are inserted into the correct locations within the ring-buffer as illustrated in figure 3.5. The ring-buffer is then 'rotated' each simulation time step – the weights accumulated in the $t + 1$ delay slot are applied to the neuron and those accumulated in the $t + 2$ delay slot are moved into the $t + 1$ slot etc.

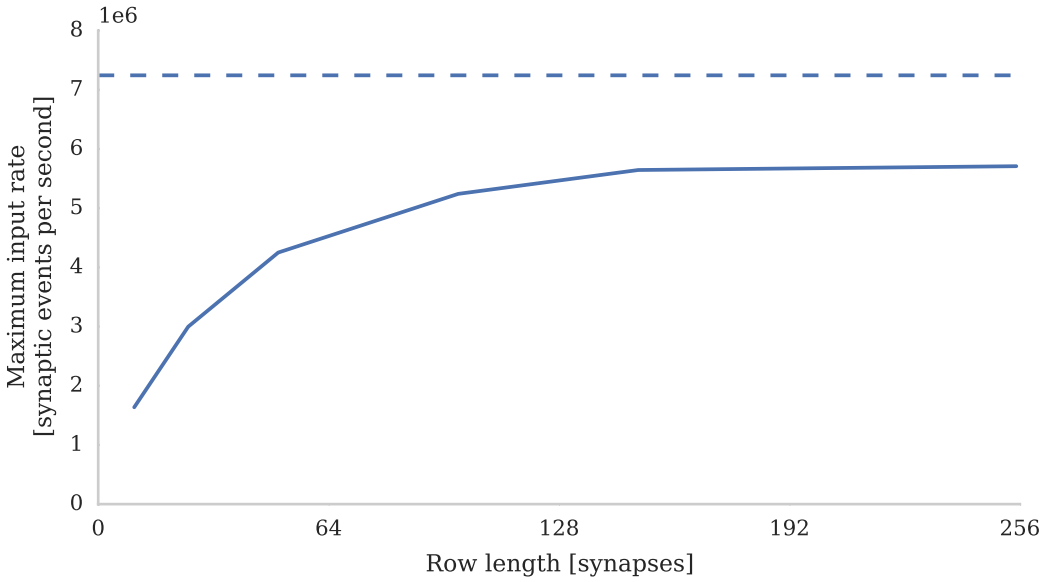


Figure 3.6: Performance of a SpiNNaker core simulating 256 neurons with varying sparseness of connectivity. Each data point represents the maximum Poisson input rate (provided by multiple 10 Hz sources) that the core can handle in real time. The dashed line illustrates the performance estimated using equation 3.14.

3.7.3 Performance

The design of SpiNNaker was based on the assumption that each processing core would be responsible for simulating 1000 spiking neurons [86]. However profiling current models shows that updating the state of a single neuron requires 187 CPU cycles rather than the 50 CPU cycles estimated by Jin *et al.* Therefore 256 neurons are typically simulated on each core. As discussed in section 2.2 each cortical neuron receives input from an average of 8000 synapses meaning that the performance of the synaptic row processing stage discussed in the previous section is critical. Profiling of the row processing code shows that processing each synapse in a row takes 21 cycles allowing us to build the following simple model of the maximum rate at which a single 200 MHz SpiNNaker core can handle incoming spikes:

Model Summary	
Populations	Neurons, stimuli
Connectivity	Probabilistic with a fixed number of postsynaptic neurons connected to each presynaptic neuron
Neuron model	Leaky integrate-and-fire with exponential-shaped synaptic current inputs
Synapse model	Current-based with exponential-shaped PSCs
Input	Independent 10 Hz Poisson spike trains

Populations		
Name	Elements	Size
Neurons	LIF	256
Stimuli	Independent 10 Hz Poisson spike trains	As described in section 3.7.3

Connectivity		
Source	Target	Weight
Stimuli	Neurons	0 nA

Neuron and synapse model	
Type	Leaky integrate-and-fire with exponential-shaped synaptic current inputs
Parameters	$g_L = 0.05\,\mu S$ leak conductance
	$C = 1\,nF$ membrane capacitance
	$V_{thresh} = -50\,mV$ threshold voltage
	$V_{reset} = -65\,mV$ reset voltage
	$V_{rest} = -65\,mV$ resting voltage
	$\tau_{syn} = 5\,ms$ synaptic time constant

Table 3.1: Model description of the benchmark network. After [85]

$$\mu_{input} \approx \frac{1}{21}\left(200 \times 10^6 - \frac{187 N_{neurons}}{dt}\right) = 7.2 \times 10^6 \tag{3.14}$$

Where the simulation time step $dt = 1$ ms and the number of neurons $N_{neurons} = 256$. Using equation 3.14 and the mean cortical firing rates of 2 Hz to 3 Hz measured by Buzsáki and Mizuseki [87] suggests that a single SpiNNaker core can easily simulate 256 neurons each with 8000 synapses.

To verify this performance estimate I developed a benchmark, described fully in table 3.1, in which a single SpiNNaker core is used to simulate a population of 256 leaky integrate-and-fire neurons. Maimon and Assad [88] suggest that, while it is not true across all areas of the neocortex, the timing of the spikes emitted by many neocortical neurons can be modelled as a Poisson process. By convention the number of spikes such models are expected to emit each second is expressed in Hz and in this benchmark the neurons are stimulated using spikes generated using multiple, independent Poisson processes with an expected firing rate of 10 Hz, generated on additional SpiNNaker cores. The maximum input spike rate that the core can handle was measured by increasing the number of inputs until the core simulating the neurons was unable to complete all processing within a 1 ms simulation time step. Figure 3.6 shows the result of this benchmark and illustrates how spike processing performance not only peaks well below that estimated by equation 3.14 but is also strongly dependent on the number of synapses in each row. This is because, beyond the 21 clock cycles spent processing each synapse, there is a significant fixed cost in: initiating the DMA transfer of the row; servicing the interrupts raised in response to the arrival of the spike and the completion of the DMA; and setting up the synapse processing loop.

Chapter 4

Spike-timing dependent plasticity

In chapter 2 I briefly discussed the highly plastic nature of synapses in the neocortex and in this chapter I will discuss synaptic plasticity in more depth. One of the earliest and most famous hypotheses on when synaptic plasticity occurs came from Donald Hebb, who postulated [89]:

> "When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased"

In this thesis I consider only the changing of the strength of *existing* connections and, in this context, Hebb's postulate indicates that connections between neurons which persistently fire at the same time will be strengthened. Neurons which persistently fire at the same time are likely to do so because they respond to similar or related stimuli.

Bliss and Lømo [90] provided the first evidence to support this hypothesis by measuring how – if two connected neurons are stimulated simulateously – the synaptic connections between them are strengthened. In networks of rate-based

neurons this behaviour has been modelled using rules such as the Bienenstock, Cooper, Munroe rule [91] and Oja's rule [92]. However the focus of this thesis is on spiking neural networks and in such networks the timings of individual spikes has been shown to encode both temporal and spatial information (section 3.1.1). Therefore in this chapter I will focus on spike-timing dependent plasticity (STDP) – a form of synaptic plasticity capable of learning such timings.

In section 4.1 I will outline some of the experimental evidence supporting STDP and discuss how STDP can be modelled in networks of spiking neurons. Then in section 4.2 I will discuss how STDP has previously been implemented on SpiNNaker and other distributed systems. Unfortunately the previous best performing SpiNNaker implementation [93] reduces the static synaptic input processing performance presented in section 3.7.3 by over 10×. This approximately corresponds to a 10× reduction in the size of model which a given SpiNNaker machine can simulate – significantly hampering our ability to simulate large plastic models of the neocortex.

I have developed a new SpiNNaker STDP implementation which has both lower algorithmic complexity than prior approaches and employs new low-level optimisations to better exploit the ARM instruction set. In section 4.3 I will discuss this implementation in depth before demonstrating, in section 4.4, that it achieves double the performance of previous solutions. This new implementation is now a key component of the SpiNNaker software developed as part of the Human Brain Project which aims to provide a common platform for running PyNN simulations on SpiNNaker, BrainScaleS and HPC platforms. The current version of this software is available from `https://github.com/SpiNNakerManchester/sPyNNaker`.

4.1 Spike-timing dependent plasticity (STDP)

Levy and Steward [94] showed that, if the experiment performed by Bliss and Lømo [90] was repeated with a delay between the stimulation of two neurons then the magnitude of the increase in weight could be reduced or even reversed. Subsequently Bi and Poo [95] measured the changes in synaptic efficacy induced in the synapses of hippocampal neurons by pairs of pre and postsynaptic spikes with different relative timings. The relationship between the magnitude of these changes and the relative timing of the pre and postsynaptic spikes is known as STDP and the data recorded by Bi and Poo suggests that it reinforces causality between the firing of the pre and postsynaptic neurons. When a presynaptic spike arrives before a postsynaptic spike is emitted the synapse is *potentiated* (strengthened). However, if a presynaptic spikes arrives after a postsynaptic spike has been emitted, the synapse is *depressed* (weakened). Furthermore the data recorded by Bi and Poo suggest that the magnitude of the changes in synaptic efficacy (Δw_{ij}) is related to the relative spike timings with the following exponential functions (figure 4.1):

$$\Delta w_{ij} = \begin{cases} F_+(w_{ij}) \exp\left(-\frac{\Delta t}{\tau_+}\right) & if\ \Delta t > 0 \\ F_-(w_{ij}) \exp\left(\frac{\Delta t}{\tau_-}\right) & if\ \Delta t \leq 0 \end{cases} \tag{4.1}$$

Where $\Delta t = t_j - t_i$ represents the relative timing of pre and postsynaptic spikes, $\tau_\pm$ defines the time constant of the exponentials and the $F_\pm$ functions define how the magnitude of the change in weight depends on the current synaptic efficacy.

Bi and Poo [95] measured how Δw_{ij} depended on the previous value of w_{ij}. In figure 4.2 I redraw their data on a double-logarithmic scale and fit straight lines to the potentiation and depression components (as suggested by Morrison *et al.* [96]).

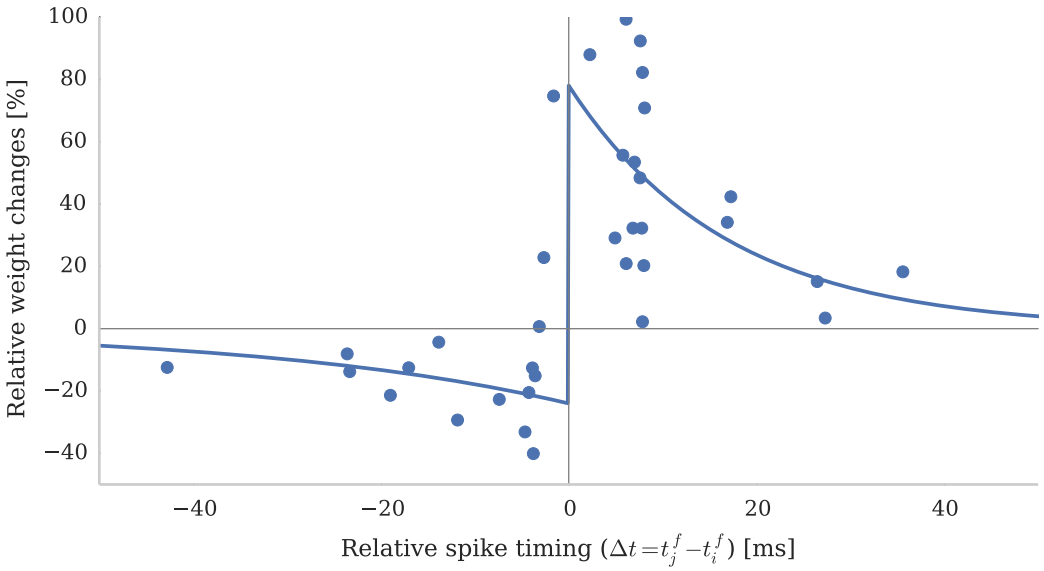


Figure 4.1: Excitatory STDP curve. Each dot represents the relative change in synaptic efficacy after 60 pairs of spikes. After [95].

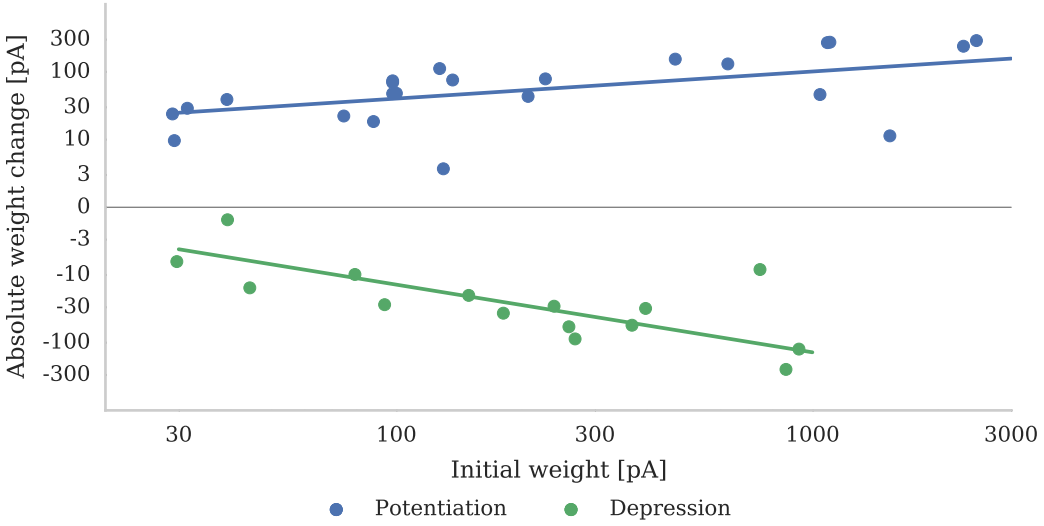


Figure 4.2: Absolute change in synaptic efficacy after 60 spike pairs. Potentiation is induced by spike pairs where the presynaptic spike precedes the postsynaptic spike by 2.3 ms to 8.3 ms. Depression is induced by spike pairs in which the postsynaptic spike precedes the presynaptic spike by 3.4 ms to 23.6 ms. Upper blue line is a linear fit to the potentiation data with slope: 0.4. Lower green line is a linear fit to the depression data with slope: −1. After [96].

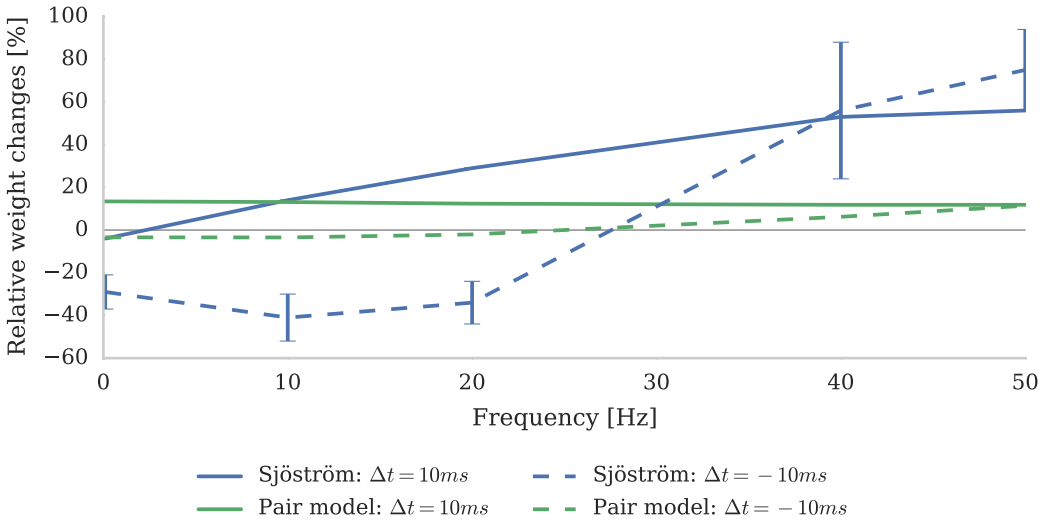


Figure 4.3: Fitting a pair-based STDP model with $\tau_+ = 16.8$ ms and $\tau_- = 33.7$ ms to data from Sjöström *et al.* [98] by minimising mean squared error fails to reproduce frequency effects. Blue lines and data-points redrawn from Sjöström *et al.* and green lines show best fit obtained by the pair-based STDP model. After [99].

The nature of the $F_\pm$ functions is indicated by the gradient of each line. Since the trend line through the depression data has a gradient of -1 it would suggest that F_- is linearly proportional to the weight. However the nature of F_+ is less clear as the trend line through the potentiation data has a gradient of 0.4. Gütig *et al.* [97] and Morrison *et al.* [96] proposed using the power law function $F_+ \propto {w_{ij}}^\mu$ to represent the magnitude of the change in weight in response to potentiation. $\mu = 0$ makes the weight update independent of the previous weight; $\mu = 1$ makes the update linearly proportional to the previous weight. With $\mu = 0.4$ the linear fit to the potentiation data recorded by Bi and Poo can be obtained.

4.1.1 Trace based models

Another common means of describing STDP rules is by using "trace variables" [96, 100, 101] which get updated when pre and postsynaptic spikes occur and represent

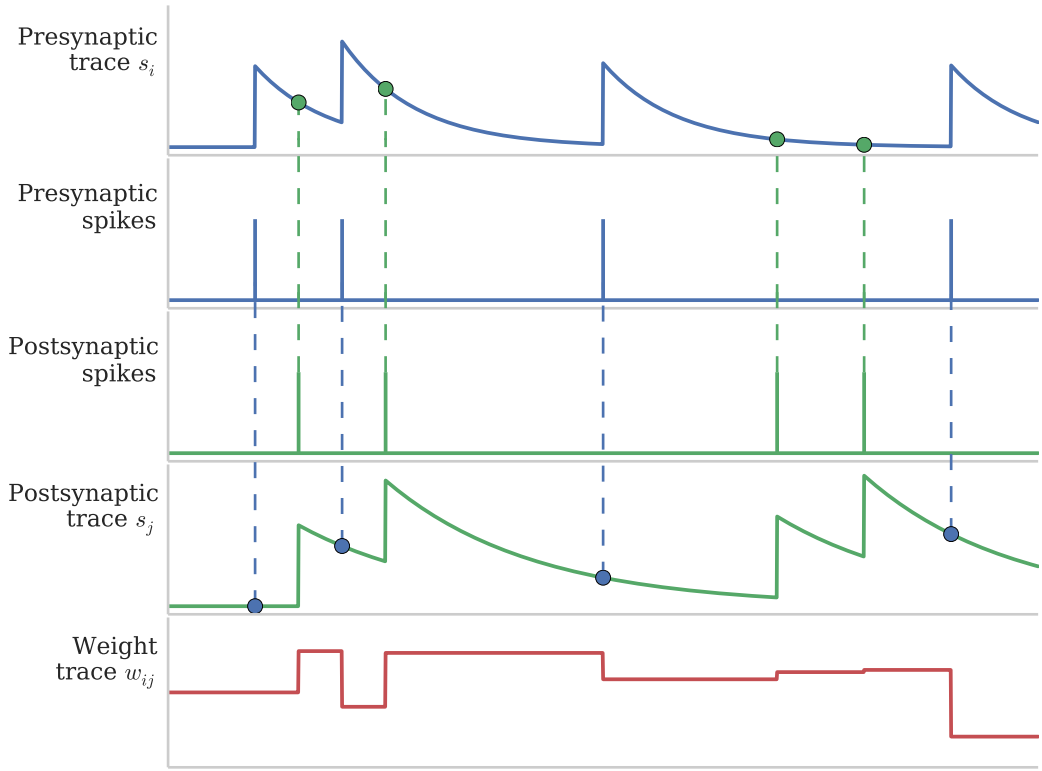


Figure 4.4: Calculation of weight updates using pair-based STDP traces. Pre and postsynaptic traces reflect activity of pre and postsynaptic spike trains. Potentiation is calculated at each postsynaptic spike time by sampling the presynaptic trace (green circle) to obtain a measure of recent presynaptic activity. Depression is calculated at each presynaptic spike time by sampling the postsynaptic trace (blue circle) to obtain a measure of recent postsynaptic activity. Weight dependence is additive. After [100].

the combined effects of the preceding pre and postsynaptic spikes. For example the interactions between individual pairs of spikes described by equation 4.1 can alternatively be modelled based on pre (s_i) and postsynaptic (s_j) trace variables:

$$\frac{ds_i}{dt} = -\frac{s_i}{\tau_+} + \sum_{t_i^f} \delta(t - t_i^f) \tag{4.2}$$

$$\frac{ds_j}{dt} = -\frac{s_j}{\tau_-} + \sum_{t_j^f} \delta(t - t_j^f) \tag{4.3}$$

Pre and postsynaptic spikes occurring at t_i^f and t_j^f respectively are modelled using Dirac delta functions (δ) and, as the top 4 panels of figure 4.4 show, the trace variables represent a low-pass filtered version of these spikes. These dynamics can be thought of as representing chemical processes. For example s_i can be viewed as a model of glutamate neurotransmitters which, having crossed the synaptic cleft from the presynaptic neuron, bind to receptors on the postsynaptic neuron and are reabsorbed with a time constant of τ_+.

As the dashed blue lines in figure 4.4 illustate, when a presynaptic spike occurs at time t_i^f, the s_j trace can be sampled to obtain the combined depression caused by the pairs made between this presynaptic spike and all preceding postsynaptic spikes. Similarly, as the dashed green lines in figure 4.4 illustate, when a postsynaptic spike occurs at time t_j^f, the s_i trace can be sampled, leading to the following equations for calculating depression (Δw_{ij}^-) and potentiation (Δw_{ij}^+):

$$\Delta w_{ij}^-(t_i^f) = F_-(w_{ij}) s_j(t_i^f) \tag{4.4}$$

$$\Delta w_{ij}^+(t_j^f) = F_+(w_{ij}) s_i(t_j^f) \tag{4.5}$$

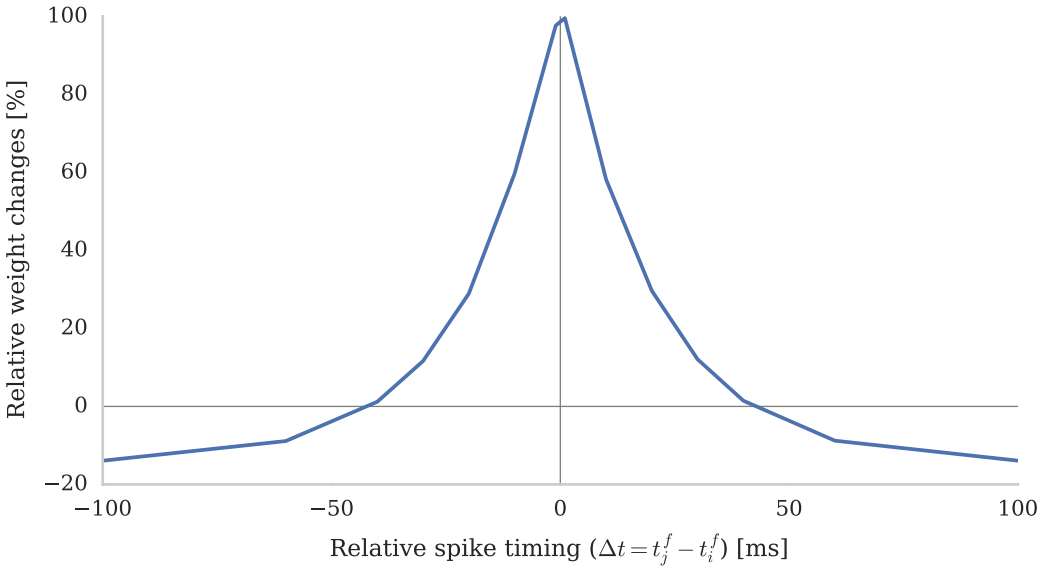


Figure 4.5: Inhibitory STDP curve. The relative change in synaptic efficacy after 60 pairs of spikes. After [102].

Bi and Poo [95] recorded the data plotted in figures 4.1 and 4.2 from rat hippocampal neurons but subsequent studies have revealed similar relationships – albeit with different time constants and polarities – in other brain areas [103]. Specifically, in the neocortex, excitatory synapses appear to exhibit STDP with similar asymmetrical kernels to hippocampal neurons whereas inhibitory synapses have a symmetrical kernel similar to that shown in figure 4.5.

While rules that consider pairs of spikes provide a good fit for the data measured by Bi and Poo they cannot account for effects seen in more recent experimental data. Sjöström *et al.* [98] stimulated cortical neurons with pairs of pre and postsynaptic spikes separated by a constant 10 ms but with between 20 ms and 10 s separating the pairs. When the time between the pairs approaches the time constants defining the temporal range of the pair-based STDP rule, spikes from neighbouring pairs begin to interact. As shown in figure 4.3 this interaction then cancels out the potentiation or depression that the original pair should have elicited.

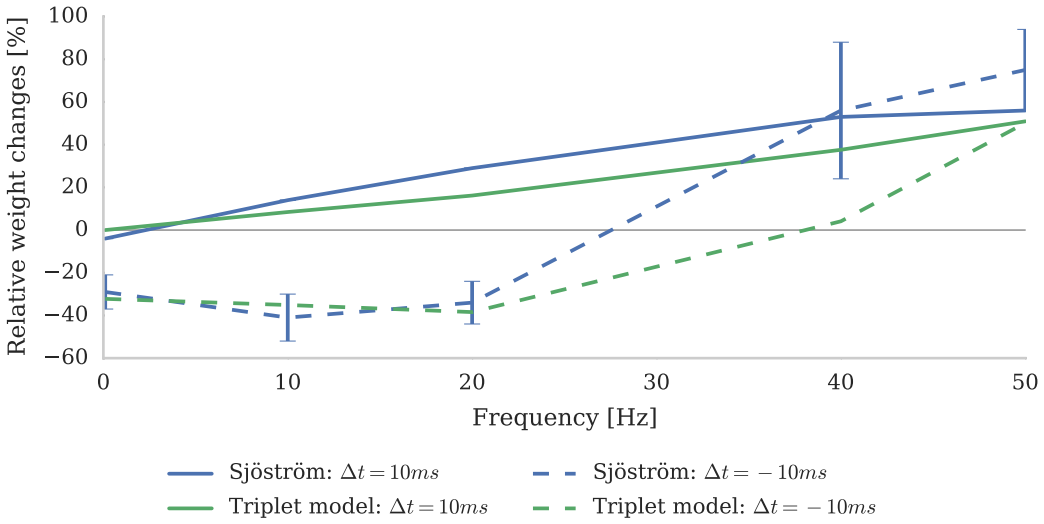


Figure 4.6: Fitting triplet STDP model with $\tau_+ = 16.8$ ms, $\tau_- = 33.7$ ms, $\tau_x = 101$ ms and $\tau_y = 125$ ms to the data recorded by Sjöström *et al.* [98] by minimising mean squared error effectively reproduces frequency effects. Blue lines and datapoints (with errors) redrawn from Sjöström *et al.* and green lines show best fit obtained by the triplet STDP model. After [99].

Several extensions to the STDP rule have been proposed which take into account the effect of multiple preceding spikes including the "triplet rule" proposed by Pfister and Gerstner [99]. In this rule the effect of earlier spikes is modelled using a second set of traces (s_i^2 and s_j^2) with longer time constants τ_x and τ_y:

$$\frac{ds_i^2}{dt} = -\frac{s_i^2}{\tau_x} + \sum_{t_i^f} \delta(t - t_i^f) \tag{4.6}$$

$$\frac{ds_j^2}{dt} = -\frac{s_j^2}{\tau_y} + \sum_{t_j^f} \delta(t - t_j^f) \tag{4.7}$$

To incorporate the effect of these traces into the weight updates Pfister and Gerstner also extended equations 4.4 and 4.5:

$$\Delta w_{ij}^{-}(t_i^f) = s_j(t_i^f)\left(A_2^{-} + A_3^{-} s_i^2(t_i^f - \epsilon)\right) \tag{4.8}$$

$$\Delta w_{ij}^{+}(t_j^f) = s_i(t_j^f)\left(A_2^{+} + A_3^{+} s_j^2(t_j^f - \epsilon)\right) \tag{4.9}$$

Where ϵ is a small positive constant used to ensure that the second set of s^2 traces is sampled just *before* the spike occurs at t_i^f or t_j^f. This rule has an explicitly additive weight dependence with the relative effect of the four traces controlled by the four free parameters A_2^+, A_2^-, A_3^+ and A_3^-. Pfister and Gerstner fitted these free parameters to the data obtained by Sjöström *et al.* [98] and, as shown in figure 4.6, demonstrated that the rule can accurately reproduce the frequency effect measured by Sjöström *et al.*

The trace-based models we have discussed so far assume that all preceding spikes can affect the magnitude of STDP weight updates. However experimental data [98] suggest that this might not be the case and that basing pair-based STDP weight updates on only the most recent spike can improve the fit of these models to experimental data. This "nearest-neighbour" spike interaction scheme can be implemented in a trace-based model by resetting the appropriate trace to 1 when a spike occurs rather than by incrementing it by 1. Pfister and Gerstner also investigated the effect of different spike interaction schemes on their triplet rule but found it had no significant effect on its fit to the data recorded by Sjöström *et al.* This suggests that alternative spike-pairing schemes may simply be another means of overcoming some of the limitations of pair-based STDP models.

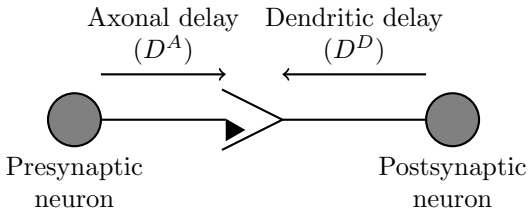


Figure 4.7: The dendritic and axonal components of synaptic delay. After [100].

4.2 Related work

Implementing the STDP rules discussed in section 4.1 in a naïve manner is relatively trivial. However, implementing them in a manner suitable for large scale simulation on a distributed system such as SpiNNaker is more difficult.

The efficient access to synaptic weights required by the event-driven synaptic processing algorithm (section 3.7.2) is facilitated by storing the synaptic matrices in a row-major format. Consequently, when a presynaptic spike arrives, weight updates (equation 4.4) can be evaluated on a row which is *contiguous* in memory. However, when a postsynaptic spike is emitted, weight updates (equation 4.3) must be evaluated on a *non-contiguous* synaptic matrix *column*. Accessing synaptic matrix columns is problematic at the hardware level as the SpiNNaker DMA controller can fetch only contiguous blocks of data. Moreover, because connectivity in the neocortex is relatively sparse, synaptic matrices are represented using a compressed sparse row structure which does *not* provide efficient access to matrix columns. To remove the need for column accesses, all of the STDP implementations presented in this section defer outgoing postsynaptic spikes to allow postsynaptic weight updates to be deferred until the next *presynaptic* spike occurs.

An additional problem regards synaptic delays. As discussed in section 3.7.2, delays are simulated on SpiNNaker by inserting synaptic weights into an input ring-buffer. However the relative timing of pre and postsynaptic spikes, and therefore the outcome of spike-timing dependent plasticity, depends on how much of

this total delay occurs in the presynaptic axon and how much in the postsynaptic dendritic tree. As shown in figure 4.7, presynaptic spikes travelling down the presynaptic axon to the synapse incur an "axonal delay" and postsynaptic spikes propagating back through the postsynaptic dendritic tree to the synapse incur a "dendritic delay".

The approaches discussed in this section differ largely in the algorithms and data structures they use to perform the deferral of postsynaptic spikes and to incorporate synaptic delays into the STDP processing. Jin *et al.* [104] were the first to implement STDP on SpiNNaker. They assumed that the whole synaptic delay was axonal implying that, as presynaptic spikes reach the synapse before this axonal delay has been applied, they too must be buffered. Jin *et al.* used a compact data structure for buffering both presynaptic and postsynaptic spikes containing the time at which the neuron last spiked and a bit field, the bits of which indicate previous spikes in a fixed window of time. Consequently, only a small amount of DTCM is required to store the deferred spikes associated with each postsynaptic neuron. However, because this approach does not use the trace-based STDP model (section 4.1) the effect of *all possible pairs* of pre and postsynaptic spikes must be calculated separately using equation 4.1. Additionally the bit field based recording of history – while compact – represents only a fixed window of time meaning that only a very small number of spikes from slow firing neurons can ever be processed.

Diehl and Cook [93] developed the first trace-based STDP implementation for SpiNNaker. To store the pre and postsynaptic traces they extended each synapse in the synaptic row to contain the values of the traces at the time of the last update. They allowed synapses to have arbitrary axonal and dendritic delays meaning that, like Jin *et al.*, they stored a history of both pre and postsynaptic spikes. However, rather than using a bit field to store this, they used a fixed-size circular buffer to

store the spike times. This data structure is not only faster to iterate over than a bit field but also holds a constant number of spikes, regardless of the firing rates of the pre and postsynaptic neurons. However, these buffers can still overflow, leading to spikes not being processed if the pre and postsynaptic firing rates are too different. For example, consider a buffer with space for ten entries being used to defer the spikes from a postsynaptic neuron firing at 10 Hz. If one of the neuron's input synapses only receives spikes (and is thus updated) at 0.1 Hz, there is insufficient buffer space for all $100 = \frac{10\,\text{Hz}}{0.1\,\text{Hz}}$ of the postsynaptic spikes that occur between the updates. Using these spike histories Diehl and Cook developed an algorithm to perform trace-based STDP updates whenever the synaptic matrix row associated with an incoming spike packet is retrieved from the SDRAM. The algorithm loops through these synapses and, for each one, iterates through the buffered pre and postsynaptic spikes in the order that they occurred since the last update (taking into account the dendritic and axonal delays). The effect of each buffered spike is then applied to the synaptic weight (using equation 4.4 for presynaptic spikes and equation 4.5 for postsynaptic spike) and the appropriate trace updated (using equation 4.2 for presynaptic spikes and equation 4.3 for postsynaptic spike). Diehl and Cook measured the performance of their approach using a benchmark network of 50 LIF neurons stimulated by a large number of 250 Hz Poisson spike sources connected with 20 % sparsity. Using this network, they showed that their approach could process 500×10^3 incoming synaptic events per second compared to the 50×10^3 achievable using the approach developed by Jin *et al.*

As discussed in section 3.7.2 there are many similarities between simulating large spiking neural networks on SpiNNaker and on other distributed computer systems – including the two problems identified at the beginning of this section. In the distributed computing space, Morrison *et al.* [96] addressed these in ways highly relevant to a SpiNNaker implementation. Although the nodes of the dis-

tributed systems they targeted do not have to access synaptic matrix rows using a DMA controller, accessing non-contiguous memory is also costly on architectures with hardware caches. Therefore postsynaptic weight updates still need to be deferred until a presynaptic spike. As each node has significantly more memory, Morrison *et al.* use a dynamic data structure to guarantee that all deferred postsynaptic spikes get processed.

Morrison *et al.* simplify the model of synaptic delay by supporting only configurations where the axonal delay is shorter than the dendritic delay. This simplification allows presynaptic spikes to be processed immediately as it guarantees that postsynaptic spikes emitted before the axonal delay has elapsed will never "overtake", and thus need to be processed before, the presynaptic spike.

This simplification means that only the time of the last presynaptic spike and the value of the presynaptic trace at that time need to be stored with each synaptic matrix row. Based on this simplification the algorithm developed by Morrison *et al.* loops through each synapse in the row and, for each one, loops through the buffered postsynaptic spikes. The effect of each buffered spike is then applied to the synaptic weight (using equation 4.5). After all of the postsynaptic spikes have been processed, the effect of the presynaptic spike that instigated the update is applied to the synaptic weight (using equation 4.4). Once all of the synapses in the row have been processed the presynaptic trace is updated (using equation 4.2).

To assess the relative algorithmic complexity of the approaches presented in this section we can consider the situation where an STDP synapse is updated based on N^{pre} presynaptic and N^{post} postsynaptic spikes. In the approach developed by Jin *et al.* [104] each pair of spikes is processed individually the complexity is $O(N^{pre}N^{post})$. However, by using a trace-based approach Diehl and Cook [93] reduced this complexity to $O(N^{pre}+N^{post})$ and Morrison *et al.* [96] further reduced this to $O(N^{post})$ by removing the need to buffer presynaptic spikes.

4.3 Implementation

The best performing SpiNNaker STDP implementation presented in the previous section was that developed by Diehl and Cook [93]. Their benchmark indicated that, using their implementation, a SpiNNaker core could process up to 500×10^3 incoming synaptic events per second – a *tenth* of the performance of the static simulator discussed in section 3.7.2. As this corresponds to a similar reduction in the size of model a given SpiNNaker machine can simulate, improving the performance of STDP is an important part of enabling large-scale neocortical simulation on SpiNNaker .

I have developed a new SpiNNaker STDP implementation based on the algorithm developed by Morrison *et al.* [96] which has lower algorithmic complexity than previous SpiNNaker implementations and employs new low-level optimisations to better exploit the ARM instruction set. In this section I present the details of this new implementation and demonstrate how it addresses the previously identified problems with distributed simulation of STDP.

4.3.1 Postsynaptic history storage

As discussed in section 4.2, Diehl and Cook used a fixed-sized data structure with space for 10 events to store the postsynaptic history. Using this system, if more than 10 postsynaptic spikes backpropagate to a synapse between updates, some will be lost. Based on the distributions of cortical neuron firing rates in rats and macaque monkeys presented by Buzsáki and Mizuseki [87] we can derive the distribution of firing rate ratios between pairs of neurons shown in figure 4.8. Based on these ratio distributions we can determine that a buffer with 10 entries will be sufficient to handle the activity at 90 % of synapses. However to prevent postsynaptic spikes being lost when the pre and postsynaptic neurons have very

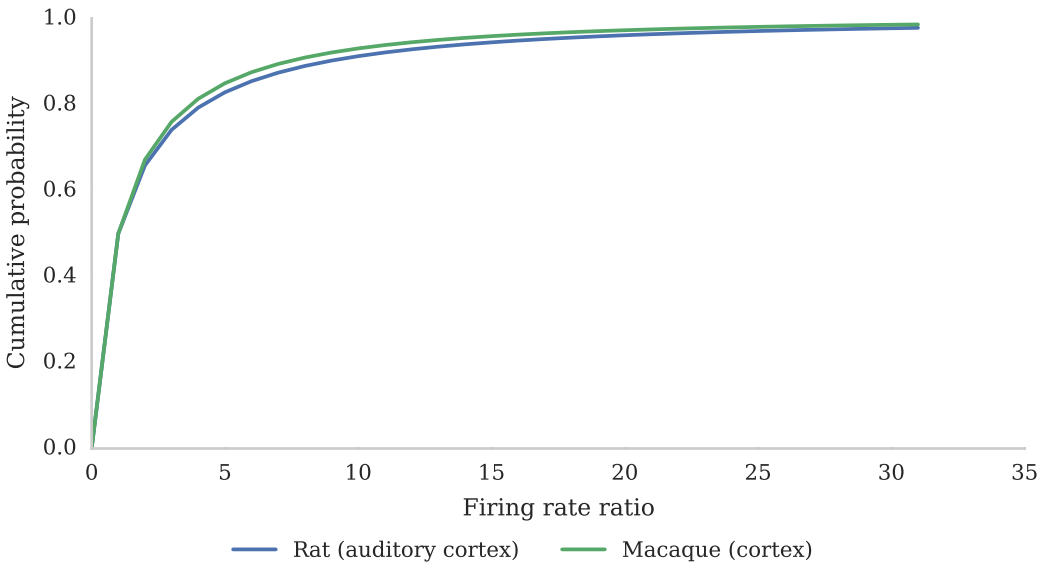


Figure 4.8: Ratio distributions of cortical firing rates. Calculated from firing rate distributions presented by Buzsáki and Mizuseki [87].

different firing rates I developed an additional mechanism I call "flushing" to force the processing of these spikes. This mechanism uses one bit in the 32 bit ID associated with each neuron to signify whether the neuron is emitting a "flush" or an actual spike event. To determine when these events should be sent, each neuron tracks its interspike interval (ISI) and, if this is *bufferSize* times longer than the ISI corresponding to the maximum firing rate of the network, a flush event is emitted.

Figure 4.9 shows the local memory requirements of postsynaptic history structures with capacity for 10 entries of different sizes. To implement STDP rules such as the triplet rule discussed in section 4.1 each entry needs to be large enough to hold not only a spike time but also two trace values. Figure 4.9 suggests that, to avoid further reductions in the number of neurons that each SpiNNaker core can simulate, each of these traces should be represented as a 16 bit value. Using 16 bit trace entries has an additional advantage as the ARM 968 CPU used by SpiNNaker includes single-cycle instructions for multiply and multiply-accumulate op-

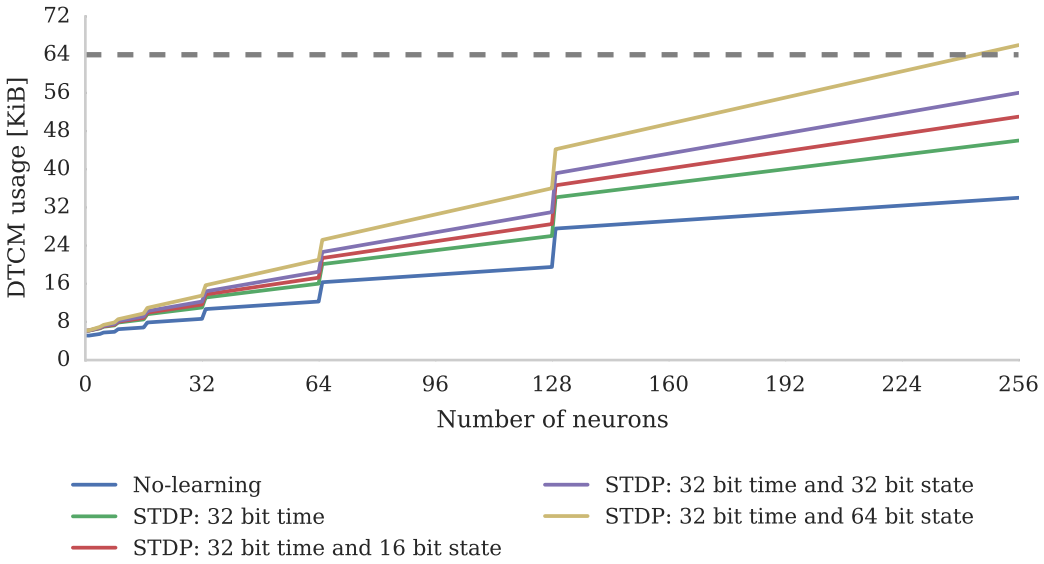


Figure 4.9: DTCM memory usage of STDP event storage schemes. Memory usage of other components based on current SpiNNaker tools. All trace-based schemes assume times are stored in a 32 bit format and traces in a 16 bit format, with two look-up tables with 256 16 bit entries providing exponential decay. The dashed horizontal line shows the maximum available DTCM.

erations on signed 16 bit integers [105]. These instructions allow additive weight updates such as $w_{ij} \leftarrow w_{ij} + s_j \exp\left(\frac{-dt}{tau}\right)$ to be performed using a single *SMLAxy* instruction and, when implementing rules such as the triplet rule that require two traces, they provide an efficient means of operating on pairs of 16 bit traces stored within a 32 bit field.

4.3.2 Fixed-point representation

As discussed in section 3.7.1 the range of fixed-point numeric representations is static. Thus the optimal representation for storing traces must be chosen ahead of time based on the maximum expected value. We can calculate this by considering

the value of a trace x with time constant τ after n spikes emitted at f Hz:

$$x(n) = \sum_{i=0}^{n} e^{-\frac{i}{\tau f}} \tag{4.10}$$

This can be rearranged into the form of a geometric sum:

$$x(n) = \sum_{i=0}^{n} \left(e^{-\frac{1}{\tau f_{max}}}\right)^{i} \tag{4.11}$$

Which has the value:

$$x(n) = \frac{1 - \left(e^{-\frac{1}{\tau f}}\right)^{n}}{1 - e^{-\frac{1}{\tau f}}} \tag{4.12}$$

Since $\left|e^{-\frac{1}{\tau f}}\right| < 1$, as $n \to \infty$ this converges to:

$$x_{max} = \frac{1}{1 - e^{-\frac{1}{\tau f}}} \tag{4.13}$$

The sustained firing rate of most neurons is constrained by the time that ion pumps take to return the neuron's membrane potential to its resting potential. This generally limits a neuron's maximum firing rate to around 100 Hz but, as Gittis *et al.* [106] discuss, there are mechanisms that can overcome this limit. For example vestibular nucleus neurons can maintain sustained firing rates of around 300 Hz. Figure 4.10 shows that – based on this worst case maximum firing rate – 4 integer bits are required to store traces with time constants in the range fitted to the data recorded by Bi and Poo [95]. Therefore a 16 bit fixed-point numeric representation with 4 integer, 11 fractional bits and a sign bit is the optimal choice for representing the traces required for pair-based STDP.

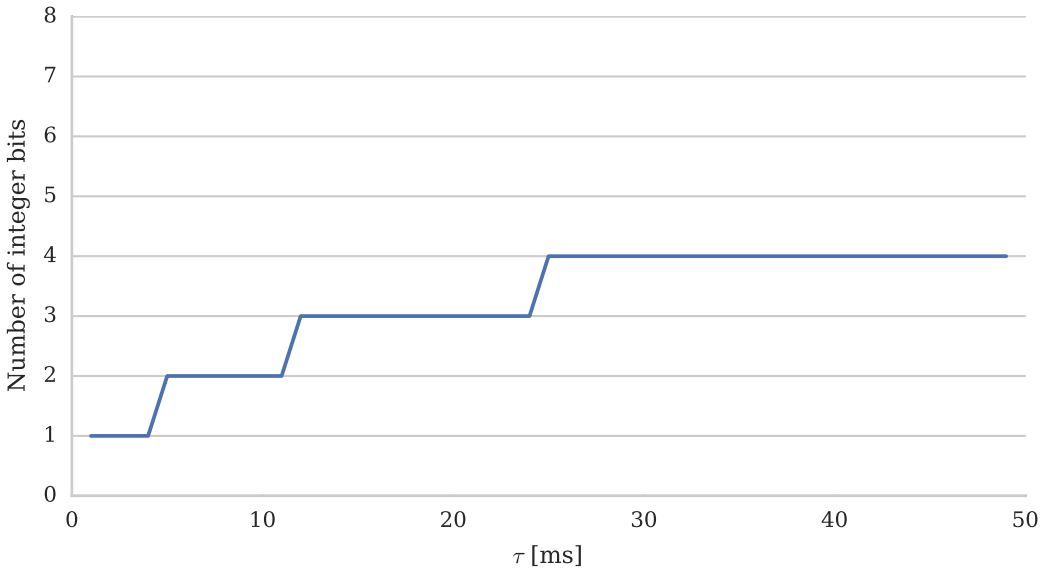


Figure 4.10: Number of integer bits required to represent traces of 300 Hz spike train with different time constants.

4.3.3 Algorithm

In the PyNN programming interface discussed in section 3.6, STDP learning rules are defined in terms of three components:

The timing dependence defines how the relative timing of the pre and postsynaptic spikes affects the magnitude of the weight update.

The weight dependence defines how the current synaptic weight affects the magnitude of the weight update (the F_+ and F_- functions discussed in section 4.1).

The voltage dependence defines how the membrane voltage of the postsynaptic neuron affects the magnitude of the weight update.

However, adding a voltage dependence to the type of event-based STDP implementation discussed in this chapter presents several challenges beyond the

scope of this thesis. Firstly the weight changes induced by several forms of voltage-based STDP [107] are continuous rather than only occurring at the time of pre or postsynaptic spikes. Secondly, assuming that the voltage takes a finite time to backpropagate from the soma to the synapse, presynaptic spikes cannot be processed immediately based on the current membrane voltage of the postsynaptic neuron.

Therefore, in this thesis, I implement only the timing and weight dependencies supported by PyNN. So as to allow users of the Human Brain Project software to not only select from the weight dependencies specified by PyNN but also easily implement their own, my implementation defines simple interfaces which timing and weight dependencies must implement. Timing dependencies must define the correct types for the pre and postsynaptic states (s_i and s_j respectively); functions to update pre and postsynaptic trace entries based on the time of a new spikes (*updatePreTrace* and *updatePostTrace* respectively) and functions to apply the effect of deferred pre and postsynaptic spikes to a synaptic weight (*applyPreSpike* and *applyPostSpike* respectively). Algorithm 1 shows an implementation of the functions required to implement pair-based STDP using this interface. The *updatePreTrace* adds the effect of a new presynaptic spike at time t to the presynaptic trace by decaying the value of s_i calculated at the time of the last spike ($t^{\text{lastSpike}}$) and adding 1 to represent the effect of the new spike (the closed-form solution to equation 4.2 between two t_i^fs). Similarly, the *applyPreSpike* function samples the postsynaptic trace by decaying the value of s_j calculated at the time of the last postsynaptic spike (t_j) (the $s_j\left(t_i^f\right)$ term of equation 4.4).

To decouple the timing and weight dependencies the *applyPreSpike* and *applyPostSpike* functions in the timing dependence call the *applyDepression* and *applyPotentiation* functions provided by the weight dependence rather than directly manipulating w_{ij} themselves. Algorithm 2 shows an implementation of *applyDe-*

Algorithm 1 Pair-based STDP timing-dependence implementation. Equivalent *updatePostTrace* and *applyPostTrace* functions are omitted for brevity.

function updatePreTrace(s_i, t, $t^{\text{lastSpike}}$)
    $\Delta t \leftarrow t - t^{\text{lastSpike}}$
    return $s_i \cdot \exp\left(-\frac{\Delta t}{tau}\right) + 1$

function applyPreSpike(w_{ij}, t, t_j, s_j)
    $\Delta t \leftarrow t - t_j$
    if $\Delta t \neq 0$ **then**
        return *applyDepression*(w_{ij}, $s_j \cdot \exp\left(-\frac{\Delta t}{tau}\right)$)
    else
        return w_{ij}

pression which performs an additive weight update.

Algorithm 2 Additive weight-dependence implementation. Equivalent *applyPotentiation* function is omitted for brevity.

function applyDepression(w_{ij}, d)
    return $w_{ij} + A^+ \cdot d$

Algorithm 3 is the result of combining the simplified delay model proposed by Morrison *et al.* [96] with the flushing mechanism and the interfaces for timing and weight dependencies discussed in this section. The algorithm begins by looping through each postsynaptic neuron (j) in the row and retrieving a list of the times (t_j) at which that neuron spiked between $t^{\text{lastUpdate}}$ and t; and its state at that time (s_j) (taking into account the dendritic (D^D) and axonal (D^A) delays associated with each synapse). The algorithm continues by looping through each postsynaptic spike and calling the *applyPostSpike* function to apply the effect of the interaction between the postsynaptic spike and the presynaptic spike that occurred at $t^{\text{lastSpike}}$ to the synapse. If the update was instigated by a presynaptic spike rather than a flush, the *applyPreSpike* function is called to apply the effect of the interaction between the presynaptic spike and the most recent postsynaptic spike to the synapse. Once all events are processed, the fully updated weight is added

to the input ring buffer. If the update was instigated by a presynaptic spike rather than a flush, after all the synapses are processed, the presynaptic state stored in the header of the row (s_i) is updated by calling the *updatePreTrace* function and $t^{\text{lastSpike}}$ and $t^{\text{lastUpdate}}$ are set to the current time. If however the update was instigated by a flush event, only $t^{\text{lastUpdate}}$ is updated to the current time, meaning that the interactions between future postsynaptic events and the last presynaptic spike will continue to be calculated correctly.

Algorithm 3 The new SpiNNaker STDP algorithm

function processRow($t, flush, t^{\text{lastUpdate}}, t^{\text{lastSpike}}, s_i, synapses$)
  for each (j, w_{ij}, d^A, d^D) **in** $synapses$ **do**
    $history \leftarrow getHistoryEntries(j, t^{\text{lastUpdate}} + d^A - d^D, t + d^A - d^D)$

    for each (t_j, s_j) **in** $history$ **do**
      $w_{ij} \leftarrow applyPostSpike(w_{ij}, t_j + d^D, t^{\text{lastSpike}} + d^A, s_i)$

    if not $flush$ **then**
      $(t_j, s_j) \leftarrow getLastHistoryEntry(t + d^A - d^D)$
      $w_{ij} \leftarrow applyPreSpike(w_{ij}, t + d^A, t_j + d^D, s_j)$
      $addWeightToRingBuffer(w_{ij}, j)$

  if not $flush$ **then**
    $s_i \leftarrow updatePreTrace(s_i, t, t^{\text{lastSpike}})$
    $t^{\text{lastSpike}} \leftarrow t$
  $t^{\text{lastUpdate}} \leftarrow t$

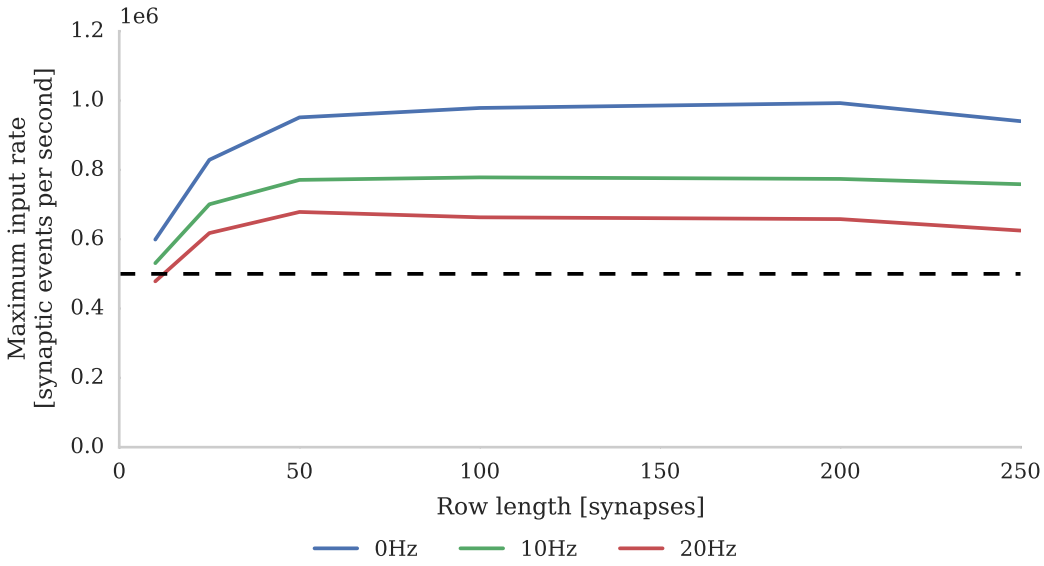


Figure 4.11: Performance of a SpiNNaker core simulating 256 neurons with STDP synapses and varying sparseness of connectivity. Each data point represents the maximum Poisson input rate (provided by multiple 10 Hz sources) that the core can handle in real time when the neurons on the core are spiking at different rates (fixed via direct current injection). The horizontal dashed line represents the performance of 500×10^3 synaptic events per second reported by Diehl and Cook [93].

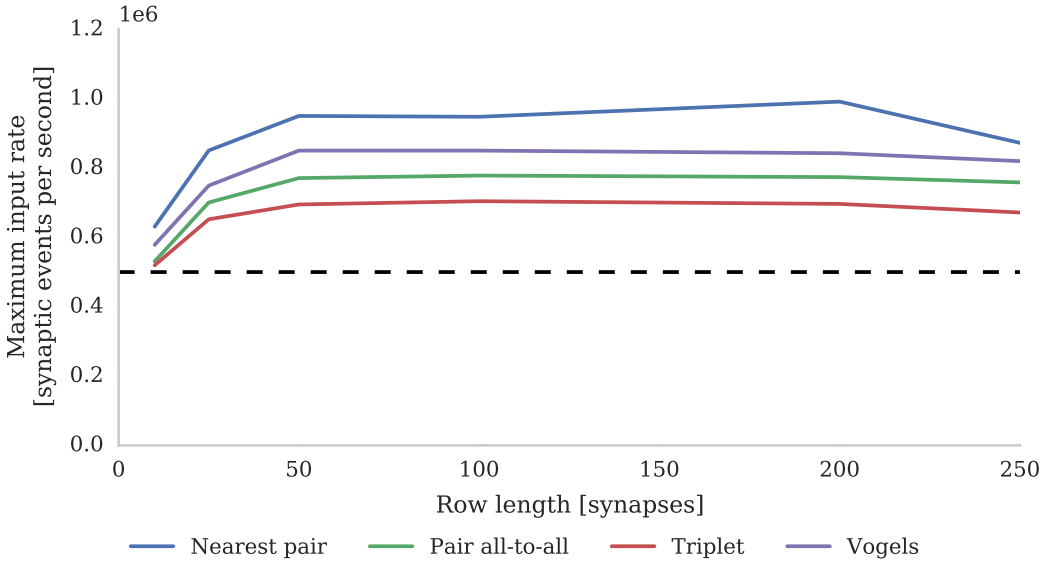


Figure 4.12: Performance of a SpiNNaker core simulating 256 neurons with different types of STDP synapse and varying sparseness of connectivity. Each data point represents the maximum Poisson input rate (provided by multiple 10 Hz sources) that the core can handle in real time when the neurons on the core are spiking at 10 Hz (fixed via direct current injection). "Vogels" refers to the inhibitory plasticity rule developed by Vogels *et al.* [102], "Triplet" to the triplet rule developed by Pfister and Gerstner [99], and "Nearest" and "all-to-all" refer to different spike-pairing schemes. The horizontal dashed line represents the performance of 500×10^3 synaptic events per second reported by Diehl and Cook [93].

4.4 Performance

Improving the performance of previous SpiNNaker STDP implementations is an important aspect of this work. Therefore in this section I will perform an in depth analysis of the performance of the new STDP implementation developed in this chapter. The cost of evaluating algorithm 3 depends on the number of events stored in *history*. Therefore I repeated the benchmark presented in section 3.7.3 on a SpiNNaker core simulating plastic synapses and used an additional DC input current to vary the postsynaptic firing rate. Figure 4.11 shows the results of this benchmark and illustrates that, probably due to the use of the simplified timing model, our algorithm out-performs the approach developed by Diehl and Cook [93] by up to 2×. Furthermore the results suggest that, due to the higher cost of updating STDP compared to static synapses, the fixed costs associated with processing a spike which we discussed in section 3.7.3 can be amortised over shorter rows.

To determine the relative cost of the different timing dependencies and spike-pairing schemes discussed in section 4.1 I also repeated the same benchmark with a nearest-neighbour spike pairing scheme, the inhibitory plasticity rule proposed by Vogels *et al.* [102] and the triplet rule proposed by Pfister and Gerstner [99]. As figure 4.12 illustrates, the performance for all four rules shows the same characteristics with sparser connectivity and therefore shorter rows resulting in lower performance. I also profiled the SpiNNaker implementations of each rule and fitted the simple models shown in table 4.1 to the performance of algorithm 3. As discussed in section 4.1 the nearest-neighbour spike-pairing is implemented by resetting each trace to 1 when a spike occurs. This not only saves a multiplication operation in each of the functions defined in algorithm 1, but also means that the values of the traces do not have to be stored or retrieved – simplifying many parts of algorithm 3. The second term of each model corresponds to the

Spike pairing	Timing Dependence	Row processing cost [cycles]
Nearest-neighbour	Spike pair	$103 + R(23P + 111)$
All-to-all	Spike pair	$125 + R(31P + 131)$
All-to-all	Inhibitory [102]	$125 + R(24P + 123)$
All-to-all	Triplet [99]	$133 + R(33P + 146)$

Table 4.1: Performance models of different STDP rules. R is the length of the synaptic matrix row (in synapses). P is the number of postsynaptic events in *history*.

cost of the *applyPostSpike* function and therefore reflects the increased complexity of the triplet rule over the other pair-based STDP rules. Similarly simpler *updatePreTrace* functions result in smaller coefficients on the first term and simpler *applyPreSpike* functions result in smaller coefficients on the third term.

4.5 Inhibitory plasticity in cortical networks

One of the simplest models of a cortical network consists of a population of excitatory neurons and a smaller population of inhibitory neurons, sparsely connected with the recurrent and reciprocal synapses shown in figure 4.13. Brunel [108] identified that these networks can operate in several well-defined regimes depending on the relative weights of the inhibitory and excitatory synapses. The asynchronous irregular regime has proved of particular interest as it matches firing rate statistics recorded in the neocortex [109] and responds rapidly to small changes in input making it an ideal substrate for computation [110]. However, it is unclear how the carefully balanced synaptic weights required to establish this regime are maintained in the brain. Vogels *et al.* [102] demonstrated that the asynchronous irregular regime can be established using an STDP rule with the type of symmetrical kernel shown in figure 4.5. I implemented this learning rule using the timing

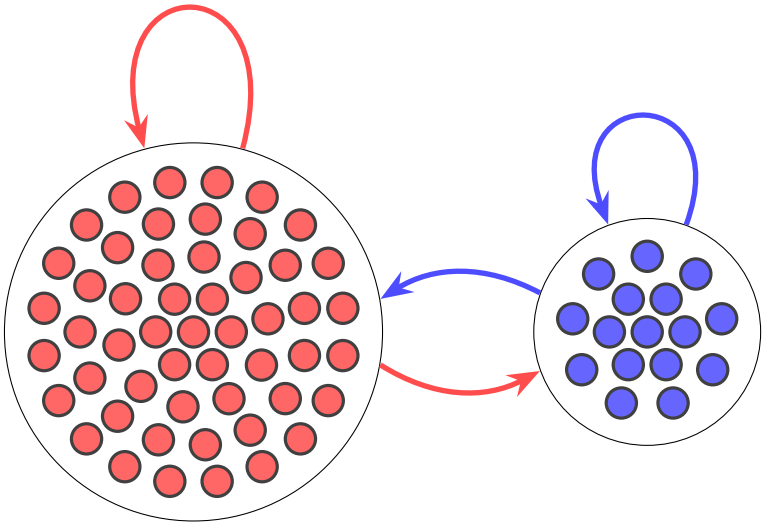


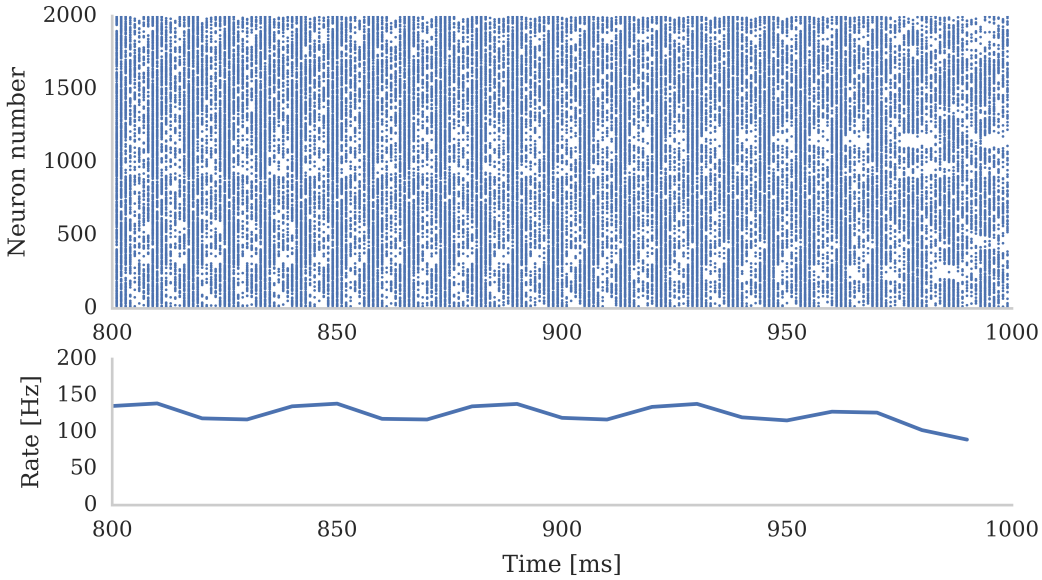
Figure 4.13: A balanced random network consisting of recurrently and reciprocally connected populations of excitatory neurons (red filled circles) and inhibitory neurons (blue filled circles). Excitatory connections are illustrated with red arrows and inhibitory connections with blue arrows.

dependence functions defined in algorithm 4 and used it to reproduce the results presented by Vogels *et al.* using a network of 2000 excitatory and 500 inhibitory neurons with the parameters listed in table 4.2.

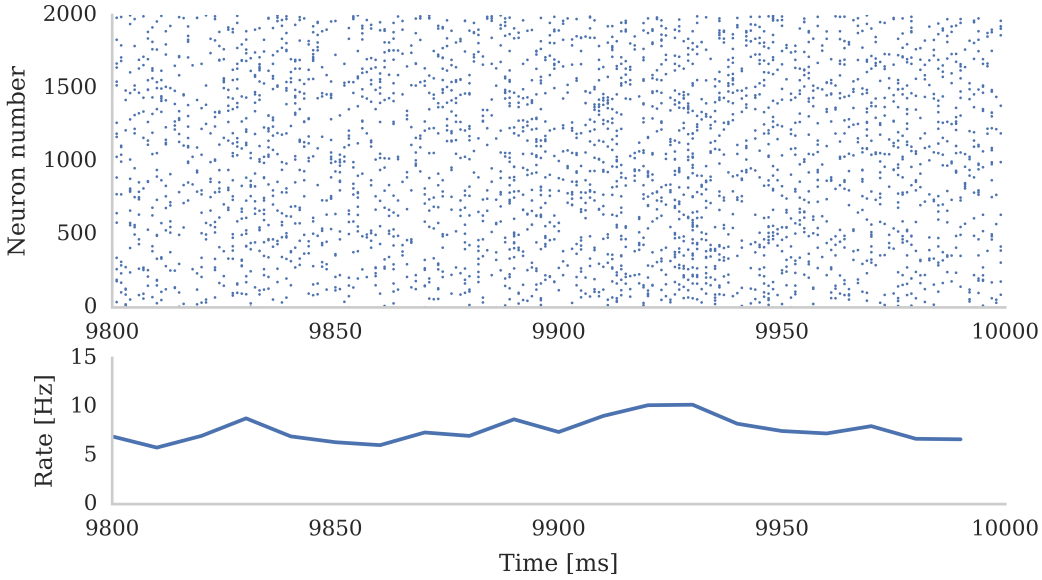
Without inhibitory plasticity the network remained in the synchronous regime shown in figure 4.14a in which neurons spiked simultaneously at high rates. However, with inhibitory plasticity enabled on the connection between the inhibitory and the excitatory populations, the neural activity quickly stabilised and, as shown in figure 4.14b, the network entered an asynchronous irregular regime in which neurons spiked at a much lower rate.

Model Summary		
Populations	Excitatory, inhibitory	
Connectivity	Probabilistic with 2 % connection probability	
Neuron model	LIF with exponential current inputs	
Plasticity	Inhibitory plasticity Vogels *et al.* [102]	
Populations		
Name	Elements	Size
Excitatory	LIF	2000
Inhibitory	LIF	500
Connectivity		
Source	Target	Weight
Excitatory	Inhibitory	0.03 nA
Excitatory	Excitatory	0.03 nA
Inhibitory	Inhibitory	0.3 nA
Inhibitory	Excitatory	0 nA
Neuron and synapse model		
Type	LIF with exponential current inputs	
Parameters	$g_L = 0.01\,\mu$S leak conductance	
	$C = 0.2$ nF membrane capacitance	
	$V_{thresh} = -50$ mV threshold voltage	
	$V_{reset} = V_{rest} = -60$ mV reset/resting voltage	
	$\tau_{syn}^{exc} = 5$ ms excitatory synaptic time constant	
	$\tau_{syn}^{inh} = 10$ ms inhibitory synaptic time constant	
Plasticity		
Type	Inhibitory plasticity Vogels *et al.* [102]	
	on Inhibitory →Excitatory synapses	
Parameters	$\rho = 0.12\,\mu$S postsynaptic target firing rate	
	$\tau = 20.0$ ms trace time constant	
	η learning rate	

Table 4.2: Model description of the inhibitory plasticity network. After [85]



(a) Without inhibitory plasticity



(b) With inhibitory plasticity

Figure 4.14: The effect of inhibitory plasticity on a balanced random network with 2000 excitatory and 500 inhibitory neurons. Without inhibitory plasticity the network is in a synchronous state with all neurons firing regularly at high rates. Inhibitory plasticity establishes the asynchronous irregular state with all neurons firing at approximately 10 Hz.

Algorithm 4 Inhibitory plasticity timing-dependence implementation. *updatePreTrace* and *updatePostTrace* functions are identical to those used by standard STDP and are therefore omitted for brevity.

function applyPreSpike(w_{ij}, t, t_j, s_j)
  $\Delta t \leftarrow t - t_j$
  return *applyPotentiation*(w_{ij}, $s_j \cdot \exp\left(-\frac{\Delta t}{tau}\right) - \alpha$)

function applyPostSpike(w_{ij}, t, t_i, s_i)
  $\Delta t \leftarrow t - t_i$
  return *applyPotentiation*(w_{ij}, $s_i \cdot \exp\left(-\frac{\Delta t}{tau}\right)$)

4.6 The effect of weight dependencies

In section 4.1 I discussed how the choice of weight dependence affects the fit of STDP models to biological data. Rubin *et al.* [111] showed that different weight dependencies also result in different equilibrium distributions of synaptic weights when neurons with a biologically-plausible number of synapses are stimulated with Poisson spike trains. Rubin *et al.* proved that a multiplicative weight dependence results in a unimodal distribution of weights whereas an additive distribution results in a bimodal distribution.

To demonstrate the flexibility of the SpiNNaker STDP implementation presented in this chapter I reproduced these results empirically using the simple PyNN model described in table 4.3. The resultant weight distributions are plotted in figure 4.15. Additive weight dependencies in PyNN specify hard upper and lower bounds and, as Rubin *et al.* predicted, the experiment using the additive weight dependence results in a weight distribution with modes centred at these bounds. Again, as Rubin *et al.* predicted, the experiment using the multiplicative weight dependence results in a unimodal weight distribution.

Model Summary		
Populations	Neurons, stimuli	
Connectivity	All-to-all	
Neuron model	LIF with exponential current inputs	
Plasticity	STDP	
Populations		
Name	Elements	Size
Neurons	LIF	1
Stimuli	Independent 15 Hz Poisson spike trains	1000
Connectivity		
Source	Target	Weight
Stimuli	Neurons	Uniformly distributed between 0 nA to 0.01 nA
Neuron and synapse model		
Type	LIF with exponential current inputs	
Parameters	$g_L = 0.017\,\mu S$ leak conductance	
	$C = 0.17\,nF$ membrane capacitance	
	$V_{thresh} = -54\,mV$ threshold voltage	
	$V_{reset} = -60\,mV$ reset voltage	
	$V_{rest} = -74\,mV$ resting voltage	
	$\tau_{syn} = 5\,ms$ synaptic time constant	
Plasticity		
Type	STDP with additive or multiplicative weight dependence	
Parameters	$A^+ = 0.01$ potentiation rate	
	$A^- = 0.0105$ depression rate	
	$\tau_+ = 20.0\,ms$ trace time constant	
	$\tau_- = 20.0\,ms$ learning rate	

Table 4.3: Model description of the synaptic weight distribution network. After [85]

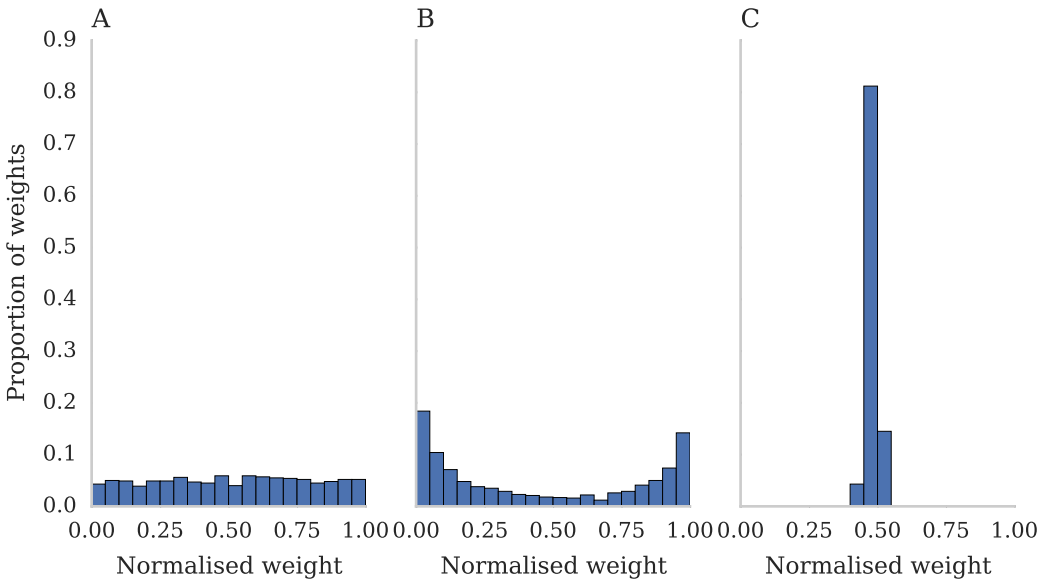


Figure 4.15: Histograms showing **A** initial uniform distribution of synaptic weights and; distribution of synaptic weights following **B** STDP with additive weight dependence and **C** STDP with multiplicative weight dependence. Simulation consists of a single integrate-and-fire neuron with 1000 independent 15 Hz Poisson spike sources providing synaptic input.

4.7 Conclusions

In this chapter I have provided some background on models of synaptic plasticity and reviewed past approaches used to simulate them on both SpiNNaker and other distributed systems. Building on these approaches I developed an entirely new SpiNNaker STDP implementation which, by both reducing the algorithmic complexity of previous approaches and employing new low-level optimisations, almost doubles the performance of previous approaches. This implementation forms a key component of the SpiNNaker software being developed as part of the Human Brain Project which aims to provide a common platform for running PyNN simulations on SpiNNaker, BrainScaleS and HPC platforms.

As demonstrated in sections 4.4 - 4.6 the modular structure of this new STDP implementation allows new weight and timing dependencies to be easily implemented. However, while any rule that relies purely on spike timings can easily be integrated in this way, there is currently some debate as to whether a dependence on postsynaptic membrane voltage [112] or its derivative [113] is in fact more fundamental than postsynaptic spike timing. As discussed in section 4.3.3, in the general case, voltage-dependent learning rules would be difficult to implement on SpiNNaker. However some voltage-dependent learning rules, such as the rule developed by Brader *et al.* [112], depend only on whether the membrane potential is above or below a threshold. In this case, as well as postsynaptic spikes, the postsynaptic history could additionally store the times at which postsynaptic neurons crossed this threshold.

Chapter 5

Bayesian Confidence Propagation Neural Networks

In chapter 4 I discussed several STDP learning rules. These rules aim to reproduce experimental data without providing any explicit theory as to their role in processing information.

An alternative approach is to start with a theoretical model of information processing and map it onto neurons and plastic synapses, taking into account restrictions such as synaptic locality and the time constants of known biological processes. Bayesian inference provides an intuitive model of how our brains internalise uncertainty about the outside world and, as such, it is a popular theoretical framework on which to base such models [114, 115]. The Bayesian Confidence Propagation Neural Networks (BCPNN) [7, 116] is a model of this type which approximates Bayesian inference using networks of neurons. Moreover, several of the synaptic plasticity phenomena discussed in chapter 4 as well as a form of "intrinsic plasticity" – which alters neuron's overall sensitivity to input – emerge from spike-based BCPNN [117].

In section 5.1 I present a derivation of the spike-based BCPNN learning rule from Bayes' theorem and this forms the basis for my implementation of the first SpiNNaker version of BCPNN which I present in section 5.2. I demonstrate the correctness of this implementation by comparing it to a previous rate-based model [118] in section 5.3 and using it to build a naïve Bayesian classifier capable of classifying the iris dataset [119] in section 5.4. Finally, in section 5.5, I build a simple model of the neocortex and show that, using BCPNN, it is capable of learning temporal sequences. I simulate this network at scales of up to 20×10^3 neurons and 51.2×10^6 plastic synapses: the largest plastic neural network ever to be simulated on neuromorphic hardware. The SpiNNaker implementation of the BCPNN synaptic plasticity rule and all the models presented in this chapter are archived by Knight [120].

Much of the material in this chapter reproduces material published by Knight *et al.* [10] in the "Anatomy and plasticity in large-scale brain models" special edition of the Frontiers in Neuroanatomy journal.

5.1 Background

In this section I derive the BCPNN learning rule from Bayesian inference and show how it can be approximated using spiking neurons. This derivation begins with a naïve Bayesian classifier network (figure 5.1). Each input layer unit represents the probability of an independent binary attribute ($P(x_i)$). Similarly output layer unit represents the probability of the input belonging to a certain class ($P(y_j)$). Using Bayes' theorem the probability of each class given the probability of each input attribute can be expressed as:

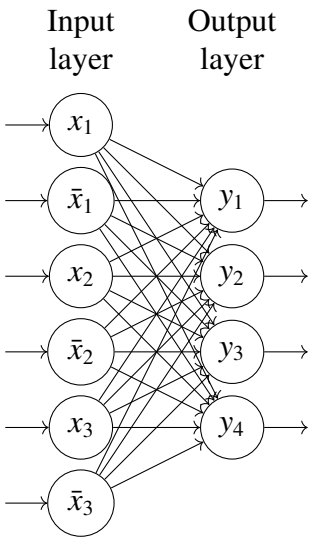


Figure 5.1: A naïve Bayesian classifier. After [7].

$$P(y_j|x_{1\ldots n}) = P(y_j)\frac{P(x_{1\ldots n}|y_j)}{P(x_{1\ldots n})} \tag{5.1}$$

Because the inputs to the classifier are assumed to be independent, we can rewrite this as:

$$P(y_j|x_{1\ldots n}) = P(y_j)\prod_{i=1}^{n}\frac{P(x_i|y_j)}{P(x_i)}s \tag{5.2}$$

Then, by taking logarithms and applying the chain rule, we obtain:

$$\log(P(y_j|x)) = \log(P(y_j)) + \sum_{i=1}^{n}\log\left(\frac{P(y_j,x_i)}{P(y_j).P(x_i)}\right) \tag{5.3}$$

Finally the terms of this equation can be separated to obtain weights (w_{ij}) for the connections between the input and output layers and biases (β_j) to apply to the

output units:

$$\beta_j = \log(P(y_j)) \tag{5.4}$$

$$w_{ij} = \log\left(\frac{P(y_j, x_i)}{P(y_j).P(x_i)}\right) \tag{5.5}$$

Lansner and Holst [7] originally trained BCPNN networks offline by counting occurrences and co-occurrences of attributes and classes and, from these, calculating probabilities and thus weights and biases. More recently Tully *et al.* [117] developed an approach for estimating $P(x_i)$, $P(y_j)$ and $P(y_j, x_i)$ online using a network of spiking neurons.

This approach begins with pre (Z_i) and postsynaptic (Z_j) traces known as the "primary traces":

$$\tau_{z_i}\frac{dZ_i}{dt} = \frac{1}{f_{max}\Delta t}\sum_{t_i^f}\delta\left(t - t_i^f\right) - Z_i \tag{5.6}$$

$$\tau_{z_j}\frac{dZ_j}{dt} = \frac{1}{f_{max}\Delta t}\sum_{t_j^f}\delta\left(t - t_j^f\right) - Z_j \tag{5.7}$$

These traces are very similar to those discussed in section 4.1 but are scaled based on the maximum allowed firing rate (f_{max}) and the simulation time step (Δt). The primary trace time constants (τ_{z_i} and τ_{z_j}) determine the time scale over which correlations can be detected and are inspired by fast biological processes. The primary traces are then fed into the "probability traces" ($P_i \approx P(x_i)$ and $P_j \approx P(y_j)$) and the probability of two neurons firing together ($P_{ij} \approx P(y_j, x_i)$) is also calculated:

$$\tau_p \frac{dP_i}{dt} = Z_i - P_i \qquad \tau_p \frac{dP_{ij}}{dt} = Z_i Z_j - P_{ij} \qquad \tau_p \frac{dP_j}{dt} = Z_j - P_j \tag{5.8}$$

While, theoretically, the time constant of these traces (τ_p) can be set to several days to match the rate of long-term cortical learning [121], this not only increases simulation times but also results in implementation issues which I will discuss in the next section. Estimated probabilities calculated using these traces are then combined to compute a postsynaptic bias membrane current (I_{β_j}) and synaptic weight between pre- and postsynaptic neurons (w_{ij}):

$$I_{\beta_j} = \beta_{gain} \log(P_j + \epsilon) \qquad w_{ij} = w_{gain}^{syn} \log \frac{P_{ij} + \epsilon^2}{(P_i + \epsilon)\left(P_j + \epsilon\right)} \tag{5.9}$$

Here β_{gain} and w_{gain}^{syn} are free parameters used to scale the dimensionless log probabilities into neuronal input currents and synaptic efficacies respectively. The lowest attainable probability estimate $\epsilon = \frac{1000}{f_{max}\tau_p}$ and the scaling used in equation 5.6 combine to establish a linear mapping from neuronal spike rates to probabilities.

5.2 Implementation

Equations 5.6 - 5.9 cannot be directly evaluated within the event-driven synaptic processing scheme outlined in section 4.3 but, as they are simple first-order linear ODEs, they can be solved to find analytic solutions for advancing the Z and P traces from the time of the last spike (t^{last}) to the current time (t). For example the presynaptic traces, Z_i and P_i can be advanced between spike times by evaluating:

$$Z_i(t_i^f) = Z_i(t^{last}) \exp\left(-\frac{\Delta t}{\tau_{z_i}}\right) + 1 \tag{5.10}$$

$$P_i(t_i^f) = P_i(t^{last}) \exp\left(-\frac{\Delta t}{\tau_p}\right) + Z_i(t^{last}) a_i \left(\exp\left(-\frac{\Delta t}{\tau_{z_i}}\right) - \exp\left(-\frac{\Delta t}{\tau_p}\right)\right) \tag{5.11}$$

With the following coefficient used for brevity:

$$a_i = \frac{1}{f_{max}\left(\tau_{z_i} - \tau_p\right)}$$

The Z_j and P_j traces can be advanced between postsynaptic spike times using a matching set of equations and P_{ij} can be advanced using the following equation:

$$P_{ij}(t) = P_{ij}(t^{last}) \exp\left(-\frac{\Delta t}{\tau_p}\right) + Z_i(t^{last}) Z_j(t^{last}) a_{ij} \left(\exp\left(-\frac{\Delta t}{\tau_{z_{ij}}}\right) - \exp\left(-\frac{\Delta t}{\tau_p}\right)\right) \tag{5.12}$$

Again with the following coefficients used for brevity:

$$\tau_{z_{ij}} = \left(\frac{1}{\tau_{z_i}} + \frac{1}{\tau_{z_j}}\right)^{-1} \qquad a_{ij} = \frac{1}{{f_{max}}^2 \left(\tau_{z_j} + \tau_{z_i}\right)\left(\tau_{z_{ij}} - \tau_p\right)}$$

Equations 5.10 - 5.12 could be used directly in the event-driven STDP algorithm developed in chapter 4. However, even compared to the triplet STDP rule implemented in chapter 4, equation 5.12 would be very costly to evaluate in the *applyPreSpike* and *applyPostSpike* functions called from algorithm 3. To reduce this complexity Vogginger *et al.* [122] converted equations 5.10 - 5.12 into a "spike-response model" [123] – A standard means of representing event-driven neural systems. Vogginger *et al.* show that, as this model consists only of linear combinations of $\exp\left(\frac{-t}{\tau_{z_i}}\right)$, $\exp\left(\frac{-t}{\tau_{z_j}}\right)$ and $\exp\left(\frac{-t}{\tau_p}\right)$, it can be re-framed into a new set of state variables: Z_i^*, Z_j^*, P_i^*, P_j^* and P_{ij}^*. The Z^* and P^* state variables have the same dynamics as the STDP trace variables discussed in section 4.1 and can therefore

be evaluated when a spike occurs at time t:

$$Z_i^*(t_i^f) = Z_i^*(t^{last}) \exp\left(-\frac{\Delta t}{\tau_{z_i}}\right) + 1 \qquad P_i^*(t_i^f) = P_i^*(t^{last}) \exp\left(-\frac{\Delta t}{\tau_p}\right) + 1 \tag{5.13}$$

These Z^* and P^* traces can now be stored in the pre and postsynaptic state (s_i and s_j) and updated both in the *updatePreTrace* function called from algorithm 3 and when postsynaptic neurons fire. P_{ij} can similarly be re-framed in terms of a new state variable which can be stored alongside the synaptic weight (w_{ij}) in each synapse. These state variables can then be updated in the *applyPreSpike* function called from algorithm 3:

$$P_{ij}^*(t_i^f) = P_{ij}^*(t^{last}) \exp\left(-\frac{\Delta t}{\tau_p}\right) + Z_j^*(t_i^f) \tag{5.14}$$

as well as in the *applyPostSpike* function also called from algorithm 3:

$$P_{ij}^*(t_j^f) = P_{ij}^*(t^{last}) \exp\left(-\frac{\Delta t}{\tau_p}\right) + Z_i^*(t_j^f) \tag{5.15}$$

The final stage of the event-based implementation is to obtain the probability trace (P_i, P_j and P_{ij}) values required to evaluate equation 5.9 from the new state variables enabling w_{ij} and β_j to be calculated:

$$P_i(t) = a_i \left(Z_i^*(t) - P_i^*(t)\right) \tag{5.16}$$

$$P_{ij}(t) = a_{ij} \left(Z_i^*(t) Z_j^*(t) - P_{ij}^*(t)\right) \tag{5.17}$$

This approach makes implementing spike-based BCPNN on SpiNNaker feasible from an algorithmic point of view but further problems arise from the need to use fixed-point arithmetic. Vogginger *et al.* [122] investigated the use of fixed-point types for BCPNN as a means of saving memory and calculated that, to match

the accuracy of a time-driven floating point implementation, a fixed-point format with 10 integer and 12 fractional bits would be required. However, not only do the individual neurons in the neocortical models which are the focus of this thesis fire at much lower rates than the entire minicolumns being considered by Vogginger *et al.*, but the ARM architecture allows only 8, 16 or 32 bit types to be natively addressed. Therefore, using the same approach discussed in section 4.3, we can re-evaluate these calculations for a SpiNNaker implementation using 16 bit signed traces. As the P^* trace has the largest time constant it will decay the slowest and therefore reach the largest value. By using equation 4.13 to calculate the maximum value of this trace we can determine that a signed fixed-point format with 6 integer and 9 fractional bits allows traces with $\tau_p < 3.17$ s to be represented if $f_{max} = 20$ Hz and with $\tau_p < 1.27$ s if $f_{max} = 50$ Hz.

The relatively large time constants of the P^* traces mean that – while more LUT entries are required before $\exp\left(\frac{-t}{\tau_p}\right)$ decays to 0 than for the STDP traces discussed in chapter 4 – this function can still be implemented using the type of simple LUTs discussed in section 3.7.1. However the $\log(x)$ function required to evaluate equation 5.9 has no convenient decaying or periodic properties and therefore needs to operate over a much larger domain. Conveniently x can be normalised into the form $x = y \times 2^n : n \in \mathbb{Z}, y \in [1, 2)$ so a LUT is only required to cover the interval $[1, 2)$ and n can be calculated using an integer log_2 operation.

5.3 Validating BCPNN learning on SpiNNaker

In this section I demonstrate that the implementation of BCPNN described in section 5.2 produces connection weights and intrinsic excitabilities comparable to those learned by previous models. To do this I use the network described in table 5.1 and the procedure developed by Tully *et al.* [117] to compare the dynamics

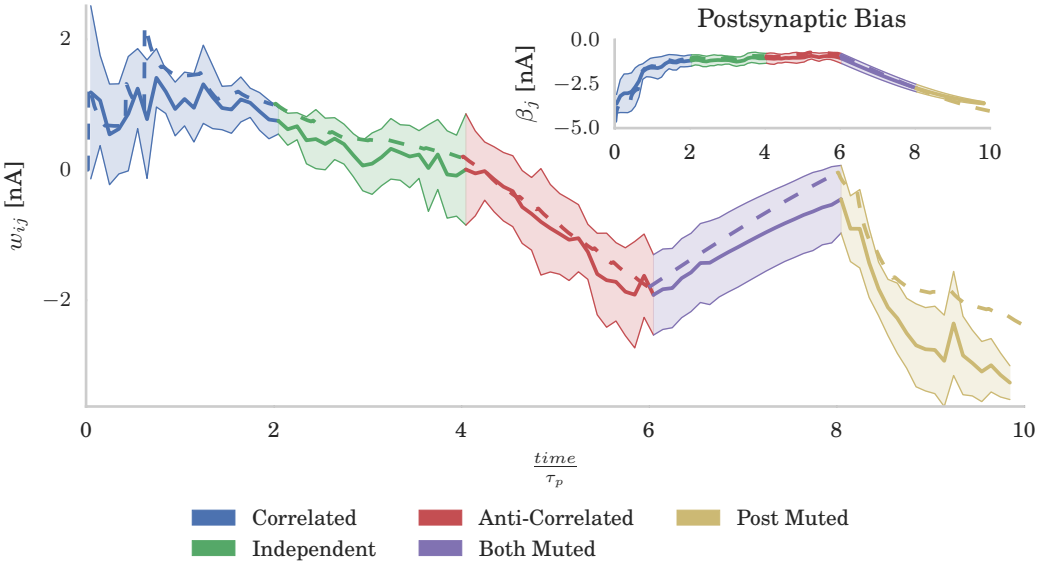


Figure 5.2: Spike-based BCPNN estimates rate-based BCPNN for different input patterns. Comparing weight and bias (inset) development under different protocols when using rate-based (dashed) and SpiNNaker (solid) versions of the learning rule. SpiNNaker simulations were repeated 10 times and averaged, with standard deviations illustrated by the shaded regions.

of a BCPNN synapse connecting two neurons modelled using both my SpiNNaker BCPNN implementation and a previous rate-based implementation [118]. I performed this comparison by presenting the neurons with five patterns of differing relative activations, each repeated for ten consecutive 200 ms trials. *Correlated* patterns meant both neurons were firing at f_{max} Hz or ϵ Hz each trial; *independent* patterns meant uniform sampling of f_{max} Hz and ϵ Hz patterns for both neurons in each trial; *anti-correlated* patterns meant one neuron fired at f_{max} Hz and the other at ϵ Hz or vice-versa in each trial; *both muted* meant both neurons fired at ϵ Hz in all trials; and *post muted* meant uniform sampling of presynaptic neuron activity while the postsynaptic neuron fired at ϵ Hz in all trials.

As figure 5.2 shows, during the presentation of patterns in which both units are firing, the weights calculated by the abstract model fall well within the standard

Model Summary		
Populations	Pre, Post, Pre stimuli, Post stimuli	
Connectivity	One-to-one	
Neuron model	LIF with exponential current inputs	
Plasticity	Spiking BCPNN	
Populations		
Name	Elements	Size
Pre, post	LIF	1
Pre stimuli, Post stimuli	independent Poisson spike trains as described in section 5.3	1
Connectivity		
Source	Target	Weight
Pre	Post	Plastic
Pre stimuli	Pre	2 nA
Post stimuli	Post	2 nA
Neuron and synapse model		
Type	LIF with exponential current inputs	
Parameters	$g_L = 0.025\,\mu S$ leak conductance	
	$C = 0.25\,nF$ membrane capacitance	
	$V_{thresh} = -55.4\,mV$ threshold voltage	
	$V_{reset} = V_{rest} = -70\,mV$ reset voltage	
	$\tau_{syn} = 2.5\,ms$ synaptic time constant	
Plasticity		
Type	Spiking BCPNN as described in this chapter	
Parameters	$f_{max} = 50\,Hz$ maximum spiking frequency	
	$\tau_{z_i} = \tau_{z_j} = 10\,ms$ primary trace time constant	
	$\tau_p = 1000\,ms$ probability trace time constant	
	$w^{syn}_{gain} = 1\,nA$ weight gain	
	$\beta_{gain} = 1\,nA$ intrinsic bias gain	

Table 5.1: Model description of the BCPNN validation network. After [85]

deviation of those calculated by the SpiNNaker model but, as units are muted, the two models begin to diverge. Further investigation into the behaviour of the individual state variables shows that this is due to the P^* term of equation 5.16 coming close to underflowing the 16 bit fixed-point format when a long time has passed since the last spike. This inaccuracy in the P^* term is then further amplified when the weights and intrinsic excitabilities are calculated using equation 5.9 as for small values of x, $\log(x)$ approaches its vertical asymptote. The standard deviations visible in figure 5.2 reflect the fact that for the spiking learning rule, the individual spikes making up the Poisson stimuli were different for each trial, but with the rate-based model there was no probabilistic element.

5.4 Demonstrating probabilistic inference

To demonstrate how the spiking BCPNN learning rule developed in this chapter can perform Bayesian inference I built a network with the architecture shown in figure 5.1 and used it to classify the classic Iris dataset [119]. Each class and input was represented by a population of 30 integrate-and-fire neurons, densely connected in the manner shown in figure 5.1. The Iris dataset contains 4 continuous measurements (sepal length, sepal width, petal length and petal width) recorded from 150 irises of 3 different species (Iris setosa, Iris virginica and Iris versicolor). To transform these continuous variables into a number of binary attributes I used the approach described by Lansner and Holst [7] and mapped each parameter to a finite mixture of Gaussian probability density functions. For example the sepal length parameter is mapped to 11 input populations with the tuning curves shown in figure 5.3. I trained the network using a training set consisting of 120 randomly ordered samples each presented for 75 ms. The four parameters associated with each sample were presented by stimulating the input units using the finite mix-

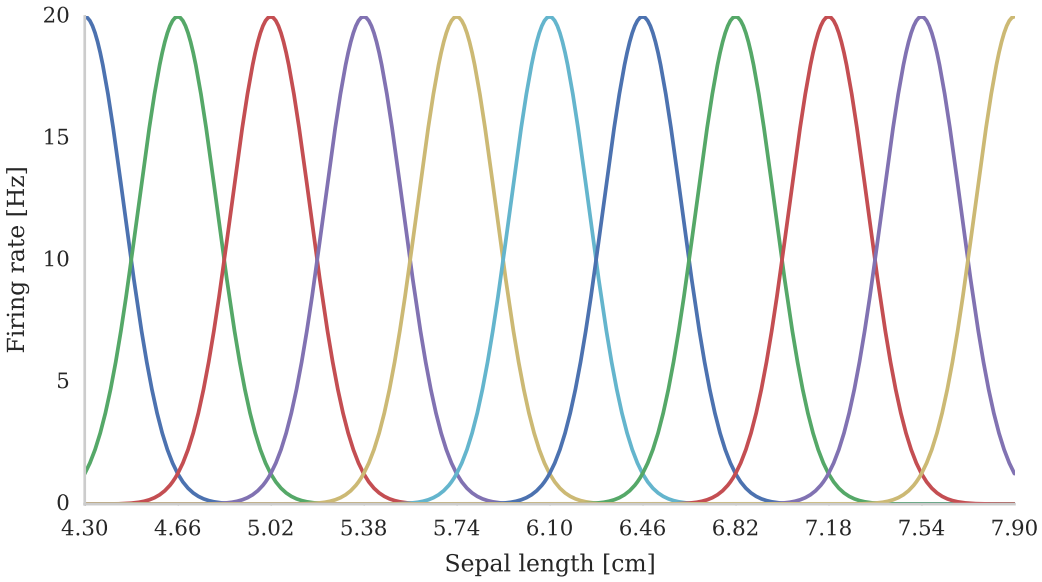


Figure 5.3: An example of a set of Gaussian component functions. Covering the interval [4.3, 7.9] of the "Sepal length" parameter of the Iris dataset [119].

ture parameter encoding and the correct class by stimulating the corresponding class population with f_{max} Hz Poisson noise. Using a testing set consisting of the remaining 30 samples I then tested the trained network by again stimulating the input units using the same finite mixture parameter encoding and recording the most highly active class population i.e. the population corresponding to the class with the highest probability. The results of this classification task are shown in the confusion matrix presented in figure 5.4 and the overall classification accuracy was 90 %. However, non-spiking naïve Bayesian classifiers typically achieve nearer 96 % [124] on this task. My analysis suggests that this poor performance is due to the training regime taking 9 s whereas τ_p is only 2 s. This means that the probability estimates calculated for earlier samples will have decayed significantly by the end of the training regime.

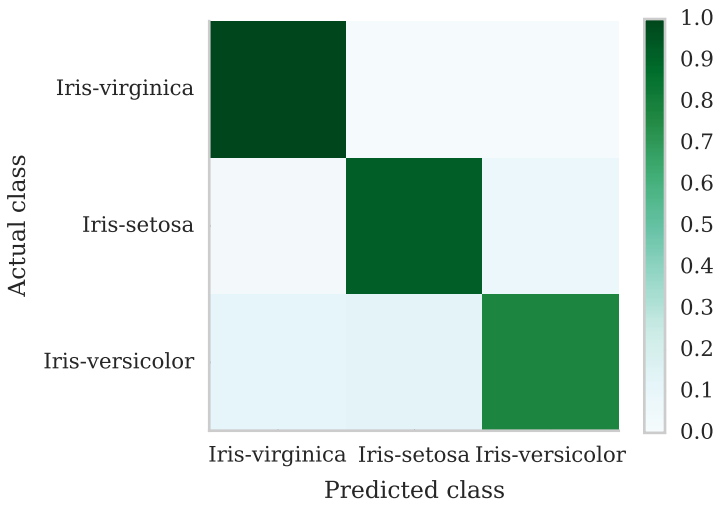


Figure 5.4: Confusion matrix from classification of Iris dataset [119] using spiking BCPNN.

5.5 Learning temporal sequences using a simplified neocortical heteroassociative memory model

One theory as to the overarching function of the neocortex comes from Lashley [8] who first suggested that all behavioural sequences were controlled by hierarchical plans. In humans, Lashley went on to say, that the production of speech is the ultimate example of this where:

> "There is a series of hierarchies of organisation; The order of vocal movements in pronouncing the word, the order of words in the sentence, the order of sentences in the paragraph, the rational order of paragraphs in a discourse."

More recently Averbeck *et al.* [9] decoded neural activity from the prefrontal cortex of macaque monkeys as they drew shapes on a screen which demonstrated several of the properties Lashley predicted. However, it remains a major challenge

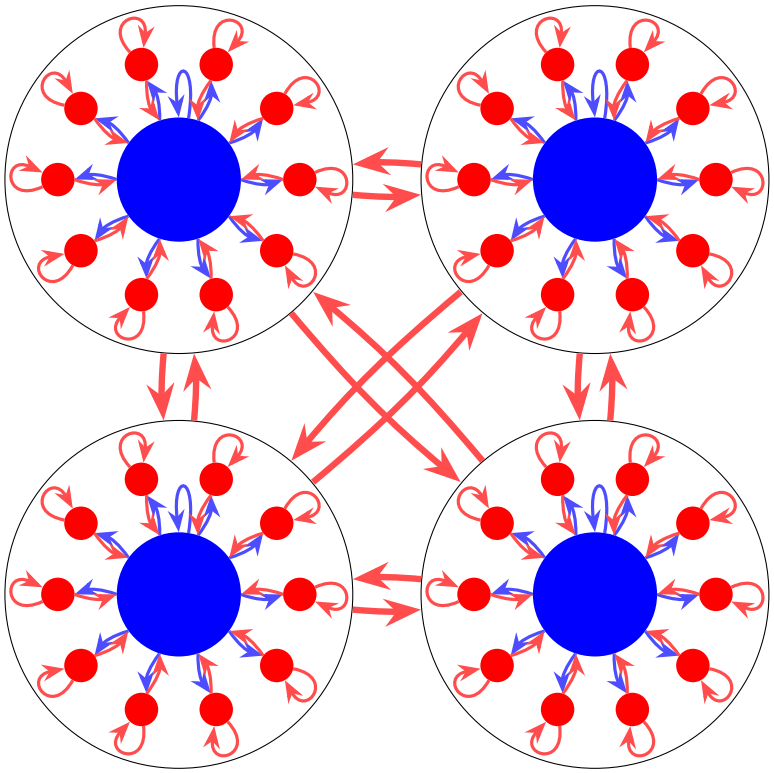


Figure 5.5: Simplified neocortical architecture with 4 macrocolumns (black outlined circles) each consisting of 10 excitatory minicolumns (red circles) and an inhibitory population (blue circle). Local inhibitory connections are indicated with thin blue arrows and local excitatory connections with thin red arrows. Global plastic excitatory connections are indicted using thick red arrows.

to learn such functionally meaningful dynamics within large-scale models using biologically plausible synaptic and neural plasticity mechanisms. In this section I use a simplified model of the neocortical architecture discussed in section 2.2 and the spiking BCPNN learning rule developed in this chapter to demonstrate one way in which this might be possible.

The network shown in figure 5.5 is inspired by the model proposed by Lundqvist *et al.* [125] and consists of N_{HC} macrocolumns arranged in a grid where each macrocolumn consists of 250 inhibitory neurons and 1000 excitatory neurons evenly divided into 10 minicolumns. The neurons in this network are modelled

using the simple LIF model presented in section 3.1.1. However, to aid the transition between sequence elements, the excitatory neurons use the simple spike-frequency adaptation mechanism proposed by Liu and Wang [126]. This mechanism generates a current I_a which is calculated as follows and subtracted from the I_{app} term of equation 3.6:

$$\tau_a \frac{dI_a}{dt} = -I_a \tag{5.18}$$

τ_a is the time constant of this adaptation process and, when the neuron emits a spike, a current of 0.15 nA is added to a. Within each macrocolumn the excitatory and inhibitory neurons are connected with the recurrent and reciprocal connections shown in figure 5.5 – enabling WTA dynamics between the minicolumns of each macrocolumn. While the strength of these synapses remains fixed, all excitatory cells in the network are also recurrently connected to each other with both fast-acting AMPA and slow-acting NMDA plastic synapses modelled using the spiking BCPNN rule. The AMPA synapses are modelled using exponential synapses with a short time constant of 5 ms and, although the NMDA synapses are also modelled using simple exponential synapses rather than a more realistic voltage-gated model, they have a longer time constant of 150 ms.

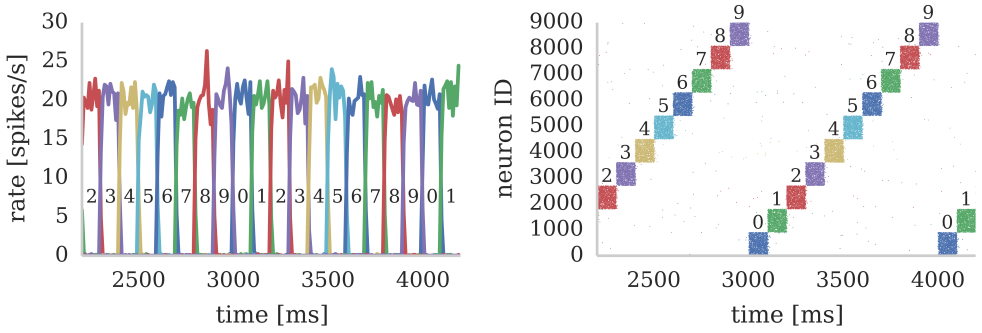
All connections in the network have distance-dependent synaptic delays based on the Euclidean distance within the grid of macrocolumns. The delay between the neurons in macrocolumns H_{xy}^{pre} and H_{xy}^{post} is therefore calculated with:

$$t_d^{H_{xy}^{pre} H_{xy}^{post}} = \frac{d_{norm} \sqrt{\left(H_x^{post} - H_x^{pre}\right)^2 + \left(H_y^{post} - H_y^{pre}\right)^2}}{V} + 1 \tag{5.19}$$

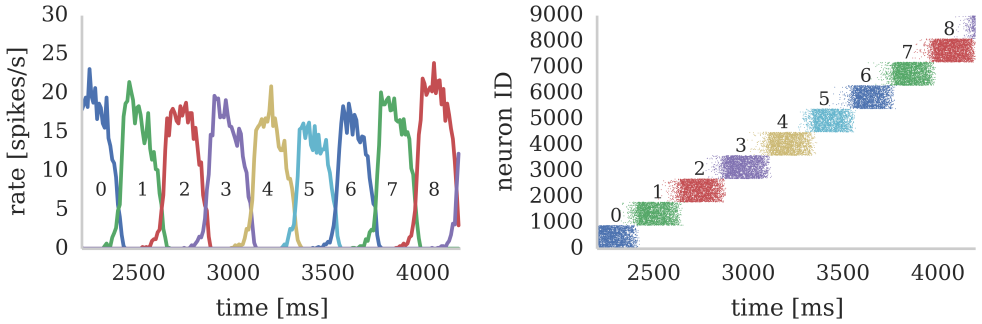
Where conduction velocity $V = 0.2\,\mathrm{mm\,ms^{-1}}$ and $d_{norm} = 0.75\,\mathrm{mm}$ mean-

ing that all connections *within* a macrocolumn have delays of 1 ms. Tully *et al.* [117] proposed that networks of this sort could learn sequences of minicolumn activation using the BCPNN learning rule. Additionally they suggested that the direction in which the network would replay these sequences could be controlled by employing asymmetrical Z_i and Z_j trace time constants. I demonstrated these abilities by employing a training regime – a subset of which is shown in figure 5.6a – in which all cells in a mutually exclusive sequence of minicolumns were repeatedly stimulated for 50 training epochs. Each minicolumn was stimulated for 100 ms, such that the neurons within it fired at an average rate of f_{max} Hz. During training I disabled the input from the plastic AMPA and NMDA synapses meaning that, while the weights were learned online, the dynamics of the network did not disturb the training regime. A recall phase followed this learning phase in which a 50 ms stimulus of f_{max} Hz was applied to all cells in the first minicolumn of the learned sequence. During both the training and recall phases I provided background input to each cell in the network from an independent 65 Hz Poisson spike source. These Poisson spike sources are simulated on SpiNNaker cores additional to those running the neural simulation algorithm described in section 3.7.2.

The training regime was able to produce the connectivity required to recall sequences in the same order in which they were presented during training as shown in figure 5.6b. Strong recurrent AMPA connectivity is learned amongst the neurons in each minicolumn as shown in figure 5.7a. This AMPA connectivity allows a stable, self-sustaining pattern of neuronal activity known as an *attractor* [127] to form if these neurons are given an initial stimulus. Transitioning between the attractor states associated with two sequence elements occurs due to the interaction between several mechanisms. Spike-frequency adaptation reduces the firing rate of the neurons within the active attractor and the NMDA connectivity shown in figure 5.7b stimulates the neurons within the attractor corresponding to the next

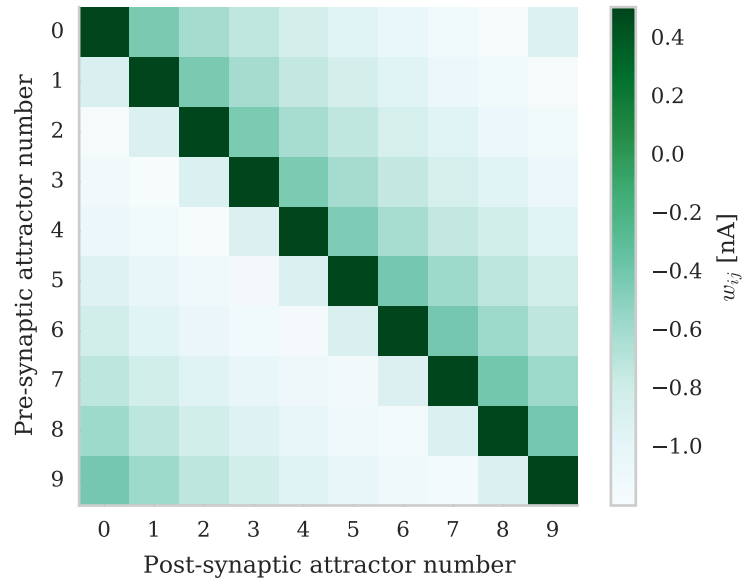


(a) Training

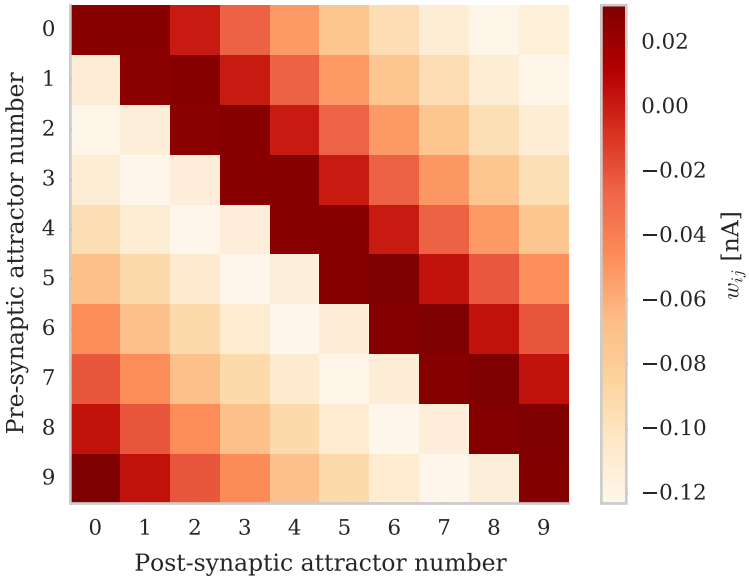


(b) Testing

Figure 5.6: Spike rasters of excitatory cells and the average firing rate within each minicolumn during subset of training and testing of temporal sequences of minicolumn activation in 9 macrocolumn modular attractor network. Active minicolumns are identified with numbers 0–9.



(a) AMPA - $\tau_{z_i} = 5$ ms and $\tau_{z_j} = 5$ ms



(b) NMDA - $\tau_{z_i} = 150$ ms and $\tau_{z_j} = 5$ ms

Figure 5.7: Average strength of learnt connections between minicolumns.

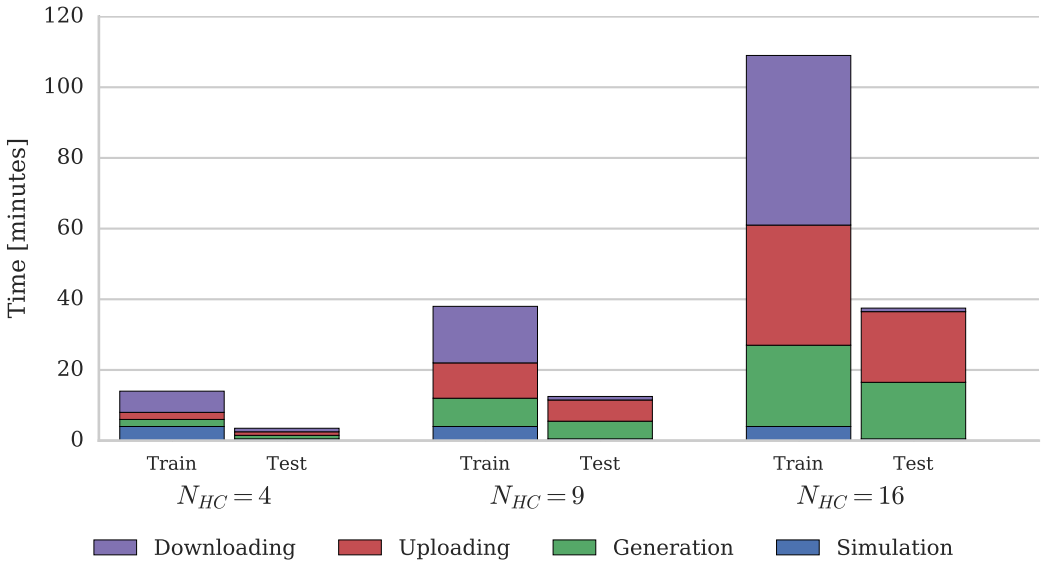


Figure 5.8: Total simulation time on SpiNNaker.

sequence element. After approximately 200 ms, the action of these mechanisms leads to the activity within the attractor corresponding to the next sequence element exceeding that within the previously active attractor. At this point, the WTA connectivity between the minicolumns acts to complete the transition, inhibiting the neurons that are not part of the newly active attractor.

Because of the modular structure of the network described in this section, this temporal sequence learning can be performed using networks of varying scales by instantiating different numbers of macrocolumns and linearly scaling the w_{gain}^{syn} parameter of the connections between them. By doing this, I investigated how the time taken to simulate the network on SpiNNaker scales with network size. Figure 5.8 shows how these times are split between the training and testing phases and how long is spent generating data on the host computer, transferring it to and from SpiNNaker, and running the simulation. So as to fit this simulation onto the single board SpiNNaker systems available at the time, I ran all of these simulations

	Simulation	SpiNNaker		Cray XC-30	
N_{HC}	time [min]	# chips	Peak power [W]	# nodes	Peak power [W]
4	17	6	6	2	938
9	50	12	12	2	938
16	146	21	21	2	938

	Simulation	SpiNNaker		Cray XC-30	
N_{HC}	time [min]	# chips	Peak power [W]	# nodes	Peak power [W]
4	9	6	6	4	1875
9	23	12	12	14[1]	6563
16	62	21	21	9	4219

Table 5.2: Comparison of power usage of modular attractor network simulations running on SpiNNaker with simulations distributed across enough compute nodes of a Cray XC-30 system to match SpiNNaker simulation time. Cray XC-30 power usage is based on the 30 kW power usage of an entire Cray XC-30 compute rack [128]. SpiNNaker power usage is based on the 1 W peak power usage of the SpiNNaker chip [81]. **Top:** SpiNNaker simulation times include downloading of learned weights and re-uploading required by current software. **Bottom:** Time taken to download learned weights, re-generate and re-upload model to SpiNNaker have been removed. [1] It is unclear why more supercomputer compute nodes are required to match the SpiNNaker simulation times when $N_{HC} = 9$ than when $N_{HC} = 16$. This is assumed to be an artefact of the different scaling properties of the two simulators, but further investigation is outside of the scope of this work.

at 0.5× real-time – allowing each core to process twice as many synaptic events within each simulation time step. This meant that the simulation time remained constant as the network grows, but the times required to generate the data and to transfer it grow significantly, meaning that when $N_{HC} = 16$ (2.0×10^4 neurons and 5.1×10^7 plastic synapses), the total simulation time is 146 min. However, the time spent in several phases of the simulation is increased by limitations of the current SpiNNaker toolchain. 84 min is spent downloading the learned weight matrices and re-uploading them for the testing – a process that is required only because the changing of parameters (in this case, whether learning is enabled or not) mid-simulation is not currently supported. Additionally, the current implementation of the algorithm outlined in section 3.7.2 only allows neurons simulated on one core to have afferent synapses with a single learning rule configuration. This means that the training regime has to be run twice with the same input spike trains, once for the AMPA synapses and once for the NMDA synapses thus doubling the time taken to simulate the training network.

Previous supercomputer simulations of modular attractor memory networks have often used more complex neuron models and connectivity [129] making simulation times difficult to compare with the SpiNNaker simulation, due to the simpler network model presented in this section. To present a better comparison, a network model with the same connectivity as the SpiNNaker model was built and simulated on a Cray XC-30 supercomputer system using NEST version 2.2 [46] with the spike-based BCPNN implementation developed by Tully *et al.* [117]. NEST does not include the adaptive neuron model described in this section so I instead used the adaptive exponential model presented in section 3.1.1 [52].

As previously discussed, SpiNNaker runs at a fixed-fraction of real-time so the NEST simulations were distributed across increasing numbers of Cray XC-30 compute nodes (each consisting of two 2.5 GHz Intel Ivy Bridge Xeon proces-

sors) until the simulation completed in the same time as those shown in figure 5.8 for my SpiNNaker simulations. Table 5.2 shows the result of both these supercomputer simulations and a second set with the time taken for the mid-simulation downloading and re-uploading of weights – currently required by the SpiNNaker software – removed. Due to this redundant step and because NEST parallelises the generation of simulation data across the compute nodes, at all three scales, the modular attractor network can be simulated using 2 compute nodes. However, if the time spent downloading and re-uploading the weights is removed, 9 compute nodes are required to match the run-time of the SpiNNaker simulation when $N_{HC} = 16$.

While a more in-depth measurement of power usage is beyond the scope of this thesis, approximate figures for the power usage of the simulations running on both systems can be derived based on the 1 W peak power usage of the SpiNNaker chip and the 30 kW power usage of a Cray XC-30 compute rack [128]. While these figures ignore the power consumed by the host computer connected to the SpiNNaker system, the power consumed by the "blower" and storage cabinets connected to the Cray XC-30 and assume that all CPUs are running at peak power usage they show that, even in the worst case, SpiNNaker uses 45× less power than the Cray XC-30 and, if the limitations of the current SpiNNaker software are addressed, this can be improved to 200×.

5.6 Conclusions

In this chapter I have demonstrated that BCPNN learning is possible on SpiNNaker. However, the implementation described in section 5.2 could also be extended to support spike-based reinforcement learning [130] by adding an extra level of *eligibility* traces with time constants between those of the primary and probability traces [117]. Eligibility traces propagate into the probability traces at a rate previously described as κ [117]. This represents a reward signal and could be used to represent basal ganglia input [131], allowing the modular attractor memory model described in section 5.5 to switch between behavioural sequences when this might be a beneficial strategy for successful task completion [132].

The original event-driven BCPNN model developed by Vogginger *et al.* [122] includes a set of E^* state variables which are used to represent the components of the spike-response model arising from the eligibility trace dynamics. Though omitted here, the SpiNNaker BCPNN implementation could be extended to include these traces at the cost of some extra computation and the memory required to store an additional 16 bit trace with each synapse and entry in the postsynaptic history structure. In section 5.3 I showed that by using a 16 bit fixed-point representation for the Z^* and P^* state variables, results comparable to previous floating-point implementations can be produced when both τ_p and f_{max} are relatively small. However, as discussed in section 5.4, τ_p restricts the number of patterns that the network can learn and additionally this approach doesn't scale to the type of model described by Fiebig and Lansner [121] where learning time constants span many orders of magnitude. In these situations, it may be necessary to use a 32 bit fixed-point representation for the P^* traces, further increasing the memory and computational cost of the learning rule.

In section 5.5 I presented simulations of a modular attractor network model with up to 16 macrocolumns, connected uniformly with 10 % connectivity. At

this scale each pyramidal cell in the network receives 4.0×10^3 afferent excitatory synapses but – if the model were scaled up to, for example, the scale of the mouse neocortex with approximately 1.6×10^7 neurons [30] – each pyramidal cell would receive 1.3×10^6 afferent synapses. However, as discussed in section 2.2, due to the "patchy" nature of long-range cortical connectivity each pyramidal cell in the neocortex receives, on average only 8.0×10^3 synapses. Additionally, while each macrocolumn in the model contains 10 minicolumns, biological macrocolumns typically have closer to 100 [27, 28]. This means that, because of the winner-take-all dynamics within each macrocolumn, while 10 % of neurons in the current model are active at any given time, only 1 % would be active in a more realistic model.

As demonstrated by the benchmarks presented in sections 3.7.3 and 4.4 the CPU load is highly dependent on the rate of incoming *synaptic events*. The combined effect of the more realistic global connectivity and sparser activity discussed in the previous paragraph would be to reduce the rate of incoming synaptic events by a factor of 5 when compared to the current model. This means that a model with more realistic connectivity could run faster than the current model on SpiNNaker.

Chapter 6

Synapse-centric simulation

The design of SpiNNaker was based on the assumption that each ARM processing core would be responsible for simulating 1000 spiking neurons. Each of these neurons was expected to have around 1000 synaptic inputs each receiving spikes at an average rate of 10 Hz [86].

However, over recent years it has become clear that larger, more realistic cortical models of the type discussed in section 2.2 are likely to break these assumptions. In section 6.1 of this chapter I will re-analyse the performance of the current SpiNNaker neural simulator and show how these, more realistic, models cannot be efficiently simulated using the approach described in section 3.7.2. Due to the flexible nature of the SpiNNaker architecture, several alternative simulators have been developed which aim to overcome some of these problems. In section 6.2 I give an overview of these alternative approaches and show that, again, in the context of more realistic models they are still likely to provide poor performance.

I solve these performance problems by developing a novel "synapse-centric" SpiNNaker simulator which I discuss in section 6.3. In sections 6.4 and 6.5 I demonstrate the improved performance of this new simulator – showing that it can *quadruple* the number of neurons with a biologically plausible number of

plastic synapses that can be simulated on a single SpiNNaker core when compared to the current SpiNNaker simulator. Finally I rerun the simulations of the neocortical model performed in the previous chapter and show that, using the new simulator, they can be performed in biological real time with two forms of simultaneously active synaptic plasticity. The version of the synapse-centric simulator used in this chapter is available from `https://github.com/project-rig/pynn_spinnaker/releases/tag/release_0_3_1`.

The work presented in this chapter largely reproduces material published by Knight and Furber [11] in the Frontiers in Neuromorphic Engineering journal.

6.1 Analysis

In sections 3.7.2 and 4.4 I showed that the synaptic input processing performance of the current SpiNNaker simulation kernel was strongly dependent on the length of the synaptic matrix rows and hence the density of the synaptic connectivity. This is clearly problematic because, as discussed in section 2.2, the connectivity of cortical networks is typically relatively sparse.

In section 2.2 I stated that cortical neurons have an average of 8000 synaptic inputs and Buzsáki and Mizuseki [87] found that the average firing rate of such neurons was at most 3 Hz. Therefore, in this section, I analyse the synaptic input processing performance of the SpiNNaker simulator based on neurons which each receive a fixed input rate of 24 kHz. I model this by stimulating a population of neurons with Poisson spike input delivered by multiple 10 Hz sources simulated on additional SpiNNaker cores. As the connectivity becomes sparser each spike source connects to fewer postsynaptic neurons via a shorter synaptic matrix row. Therefore more input spikes, and hence synaptic matrix rows need to be processed to handle the same total input spike rate. As figure 6.1b shows, this leads to synap-

tic input processing performance dropping to only 60 % of the peak performance even at the maximum biological connection density of 20 % [35]. As discussed in section 3.7.2 this occurs because, beyond the cost of processing each synapse, there is a significant fixed cost in processing each row. Furthermore, to maintain the desired input rate, the only way to counteract the decreasing performance is to further reduce the number of neurons simulated on each core which further reduces the length of the synaptic matrix rows and thus exacerbates the problem.

To increase temporal accuracy [133] or improve the numerical precision with which the differential equations used to model each neuron are solved [82], it can be necessary to simulate the time-driven components of the SpiNNaker simulation on a shorter time step such as 0.1 ms. Synaptic processing is event-driven and therefore reducing the simulation time step will have no direct effect on its computational cost. However if the simulation time step is reduced from 1 ms to 0.1 ms, so as to leave the same number of CPU cycles available for synaptic processing, 10× fewer neurons can be simulated on each core leading to row lengths also being reduced by 10×. In this situation, as figure 6.1a shows, even with 100 % connectivity the row length is sub-optimal so that only 70 neurons can be simulated on each core. Furthermore, as the connectivity becomes sparser, performance drops to the point where, at 10 % connectivity, it is impossible to simulate the benchmark in real time.

I continued this analysis of the synaptic input processing performance by measuring the performance of pair-based STDP synapses with an additive weight dependence using the same benchmark network. Figure 6.2 shows that, much like the static synaptic processing performance discussed earlier in this section, performance drops to only around 30 % of the peak performance at 20 % connectivity due to very short row lengths.

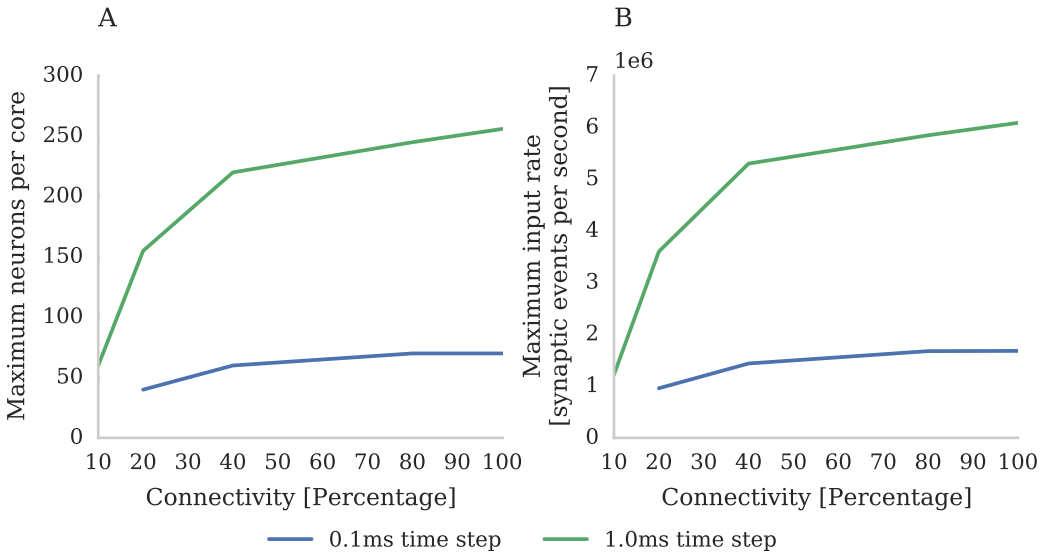


Figure 6.1: Static synaptic processing performance of a single SpiNNaker core simulating neurons using simulation time steps of 1 ms and 0.1 ms. Each neuron receives 24 kHz of synaptic input from multiple 10 Hz Poisson spike sources, connected with varying degrees of connection sparsity. With a simulation time step of 0.1 ms it was impossible to run simulations with connectivity sparser than 20 % in real time. **(A)** Performance in terms of the maximum number of these neurons that can be simulated on each core. **(B)** Performance in terms of the raw synaptic event processing performance of each core.

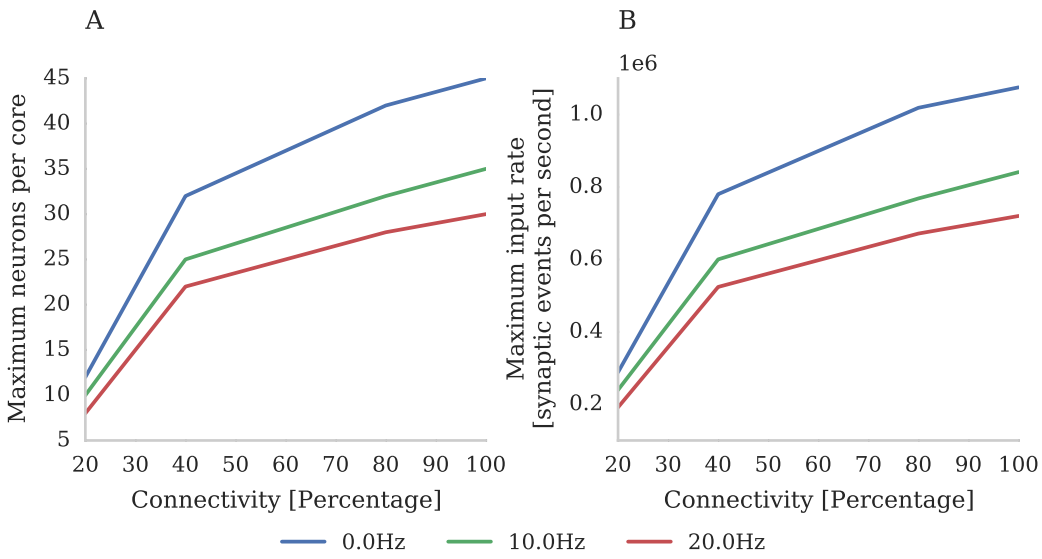


Figure 6.2: STDP synaptic processing performance of a single SpiNNaker core simulating neurons with three different postsynaptic firing rates. Each neuron receives 24 kHz of synaptic input from multiple 10 Hz Poisson spike sources, connected with varying degrees of connection sparsity. **(A)** Performance in terms of maximum number of these neurons that can be simulated on each core. **(B)** Performance in terms of raw synaptic event processing performance of each core.

6.2 Related work

Galluppi *et al.* [134] developed a very different approach for simulating synaptic plasticity on SpiNNaker compared to the event-driven approaches discussed in section 4.2. They simulated neurons and their synaptic inputs using the standard approach described in section 3.7.2 but used extra cores to simulate plasticity using a time-driven approach. These "plasticity cores" operated on a relatively slow time step of 128 ms within which they read back the entire synaptic matrix row-by-row. Then, based on a record of pre and postsynaptic activity recorded into shared memory during the previous 128 ms by the "neuron core", the plasticity core updated the synaptic weights. Galluppi *et al.* reported that, with each neuron core simulating 100 neurons, this system could perform STDP synaptic processing at rates of up to 1.5×10^6 synaptic events per second per core. However, this was based on a benchmark in which the population of neurons being simulated received input from just 195 densely connected, high frequency Poisson inputs meaning that just 195 rows needed to be processed within each 128 ms plasticity time step. If, however, we consider the model of cortical connectivity described in section 2.2 where each neuron has 8000 sparsely connected inputs, even if the connection density is 20 %, the synaptic matrix will contain 40 000 rows. If we assume the lowest mean cortical firing rate of around 2 Hz measured by Buzsáki and Mizuseki [87], each row will contain approximately 20 synapses (based on the 100 neurons per core used in the benchmark) and each row update will have to process an average of 5 postsynaptic events during each 128 ms plasticity time step. As each of the updates performed by the plasticity cores uses a trace-based approach similar to algorithm 3 we can estimate the cost of each resultant update based on the performance model presented in section 4.4. This model suggests that updating each row will take around 2800 CPU cycles meaning that, as a SpiNNaker core has 256×10^5 clock cycles available within each 128 ms plas-

ticity time step, each plasticity core would be able to update approximately 9100 rows within this time. Therefore, the updating of all 40 000 rows would need to be distributed amongst 5 plasticity cores. This would result in a per-core synaptic processing performance of just 350×10^3 synaptic events per second, only a marginal improvement over the 289×10^3 synaptic events per second achieved by the current approach in the benchmarks presented in the previous section.

Lagorce *et al.* [133] recently presented an alternative means of simulating neurons with 1 µs temporal accuracy on SpiNNaker using an event-driven neuron model. While some sensory neurons [135] may require this degree of temporal accuracy, in cortical networks of the type considered in this thesis, spike timings are typically only synchronised to within several ms [136]. Additionally, as discussed in section 3.2, only a small subset of neuron models can be simulated in an event-driven manner and cortical connectivity means that simulating neurons using an event-driven approach is likely to have little performance benefit. For these reasons, in the rest of this chapter, only time-driven neural models will be considered.

6.3 Implementation

In section 6.1 I identified two main problems with the current approach to mapping large, highly-connected spiking neural networks to SpiNNaker.

1. Synaptic processing performance decreases as connectivity becomes sparser due to shorter synaptic matrix rows over which to amortize the fixed costs of servicing interrupts, initiating the DMA transfer of the synaptic matrix row etc.

2. The only way to reduce the load on a single SpiNNaker core and thus allow neurons with a given synaptic input rate to be simulated in real time is to

reduce the number of neurons being simulated on the core, exacerbating the first problem.

In this section I present a novel solution to mapping spiking neural networks with both plastic and static synapses to SpiNNaker which alleviates both of these problems. The key intuition behind this approach is that, if the synaptic matrix is split in a row-wise manner over multiple cores rather than column-wise with the neurons, row lengths can be kept as long as local memory restrictions allow and are unaffected by dividing the synapses amongst multiple cores. As shown in figure 6.3 I achieve this by using separate cores to simulate the neurons and their afferent synapses. The afferent synapses associated with the population are split between one or more *synapse processors* based on the following criteria:

1. By synapse type, meaning that each synapse processor needs to have only sufficient local memory for a single input ring-buffer and different synaptic plasticity rules can be simulated on separate cores.

2. Postsynaptically (vertically) based on the local memory requirements of the ring-buffer structure and, if the core is simulating plastic synapses, the postsynaptic history structure required for the plasticity algorithm (as discussed in section 4.3).

3. Presynaptically (horizontally) based on an estimate of the presynaptic processing cost derived from the firing rate of the presynaptic neurons and their connectivity.

The local memory requirements of the input ring-buffer limits each synapse processor to simulating the static synapses associated with 1024 postsynaptic neurons. The extra local memory required for the postsynaptic history structure, discussed in section 4.3, limits synapse processors to simulating the STDP synapses

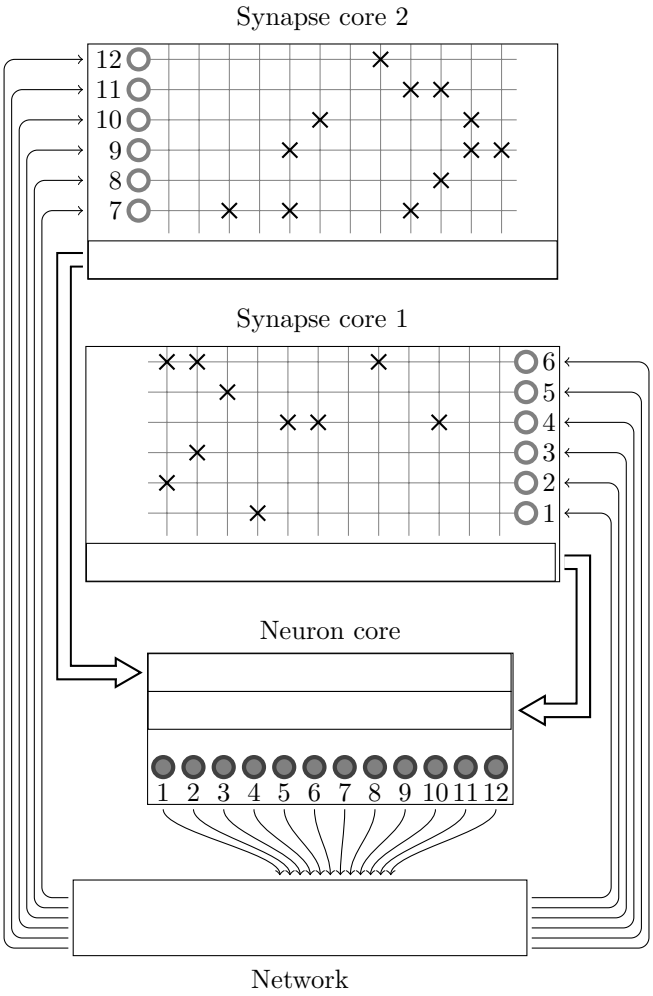


Figure 6.3: The example network used in figure 3.4 is distributed amongst three SpiNNaker cores using the synapse-centric approach. The neuron core is responsible for simulating all 12 neurons (filled circles). The synaptic matrix is split horizontally with the rows associated with the presynaptic neurons (non-filled circles) distributed between two synapse cores. Double arrows indicate how input currents or conductances are transferred from the synapse processors to the neuron processors through shared memory.

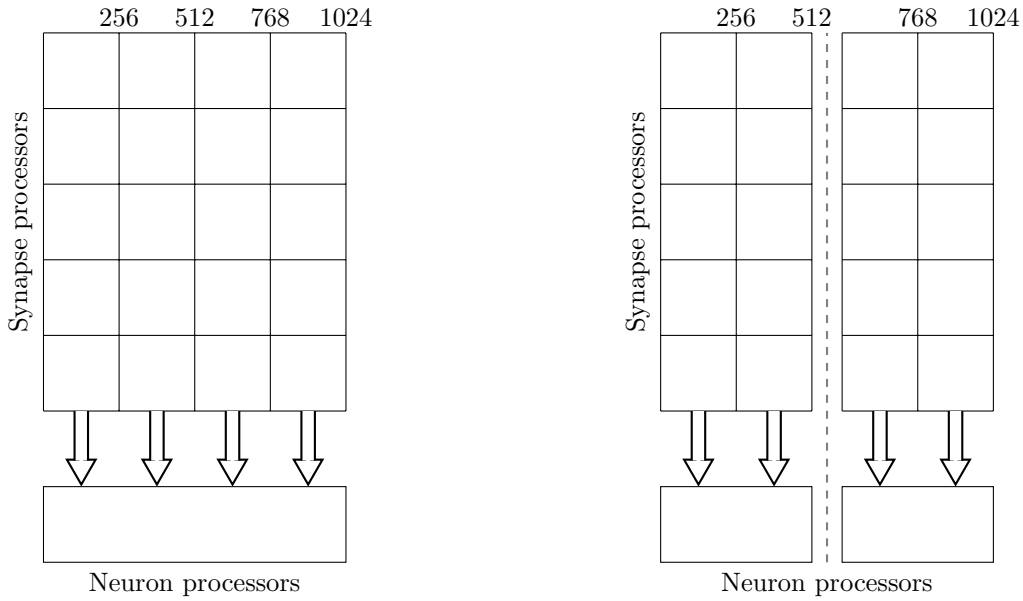
associated with 512 postsynaptic neurons. At the beginning of each simulation time step the synapse processors initiate a DMA transfer to write the input current or conductance accumulated in the ring-buffer (which in the current approach would be passed directly to the neuron model) to a buffer located in the external SDRAM.

The time-driven simulation of the neurons is split amongst *neuron processors* until memory and real time CPU constraints are met. Without having also to simulate the afferent synapses, each neuron processor can simulate many more neurons than is possible using the current approach. For example 1024 LIF neurons with exponential synapses simulating on a 1 ms time step can be simulated on a single core.

At the beginning of each simulation time step each neuron processor initiates a series of DMA reads to fetch the buffers containing the input currents or conductances written by its associated synapse processors. The current or conductance inputs associated with each of the neuron model's receptors are then summed together and passed to the neuron model.

Although the postsynaptic splitting of neurons and synapses can be independent, because the neuron and synapse processors communicate through shared memory buffers only accessible to the 16 cores on the same SpiNNaker chip, this is somewhat restricted.

For example, if we consider a population of simple LIF neurons (1024 of which can be simulated on a single core) with complex plastic synapses whose local memory requirements mean that they must be split postsynaptically at 256 neurons. If, presynaptically, 5 synapse processors are required to handle the input to these 256 neurons then, as figure 6.4a illustrates, 21 cores would need to access the same shared memory buffer – more than are available on the SpiNNaker chip. The solution to this problem is to reduce the number of neurons simulated on



(a) Splitting with 1024 neurons per neuron processor – Does not fit on a single SpiNNaker chip

(b) Splitting with 512 neurons per neuron processor – Fits on two SpiNNaker chips

Figure 6.4: Limitations on synapse-centric splitting of neurons and synapses. Double arrows indicate how input currents or conductances are transferred from the synapse processors to the neuron processors through shared memory.

each neuron processor to 512 meaning that, as figure 6.4b illustrates, only 9 cores would need to access the same shared memory buffer.

As well as the inputs they receive from other neurons in the network, neurons in cortical models are often kept in the asynchronous irregular regime by a source of background noise. This background input often takes the form of independent Poisson spike trains and, when using the approach discussed in section 3.7.2, these are delivered to the neurons using the SpiNNaker interconnect network. However, the mechanism for providing input to a neuron processor through external memory buffers can be re-used to allow the background input to be delivered from *current input processors* directly to the neuron processors. The current input processors generate a Poisson spike vector every time step, multiply it by a weight vector to convert the spikes into current or conductance values, and write the resulting vector to the external memory buffers.

The approach described in section 4.3 would also be difficult to extend to allow populations of neurons to have multiple learning rules on their afferent synapses. This would require the postsynaptic history structure to be extended to include postsynaptic state (s_j) for each learning rule – adding to its already considerable memory requirements with each additional learning rule. Algorithm 3 would also have to be extended to select the correct learning rule for each synapse and call the appropriate *applyPostSpike* and *applyPreSpike* functions – increasing the cost of this, performance-critical, algorithm. However, supporting multiple learning rules is trivial when using the synapse-centric approach; additional synapse processors can simply be instantiated to simulate each required synapse type.

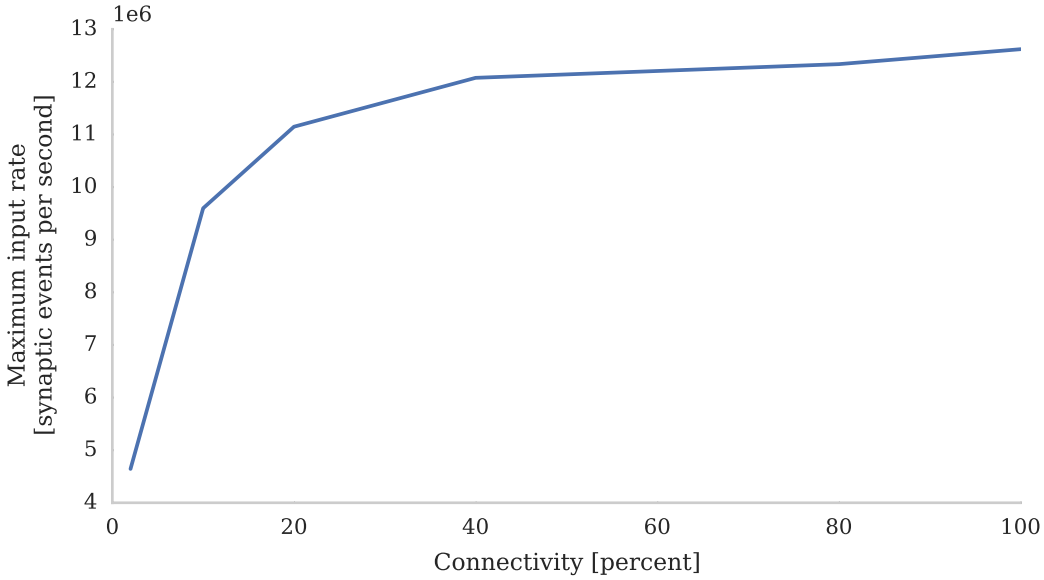


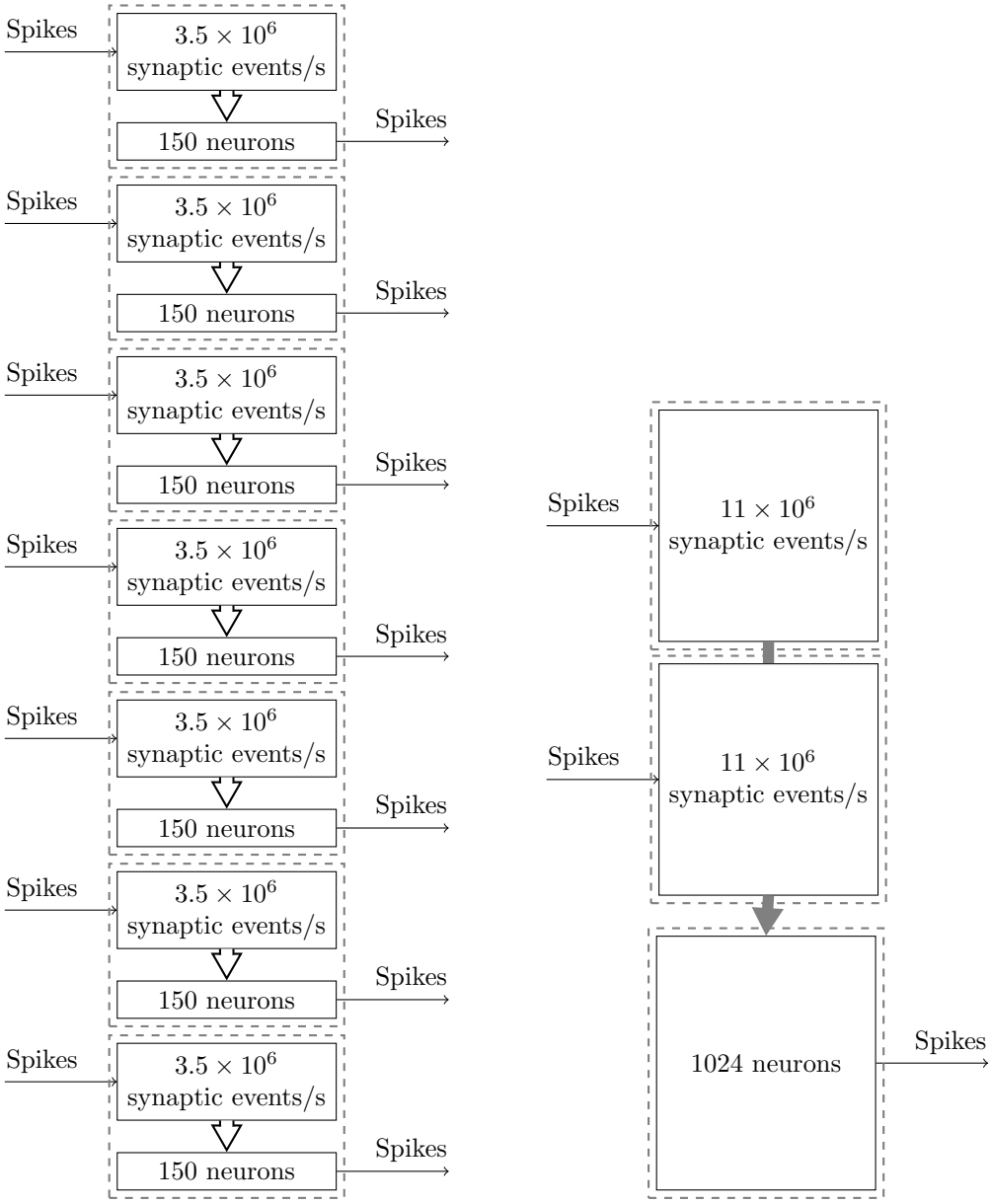
Figure 6.5: Performance of a synapse processor simulating the afferent static synapses associated with 1024 neurons. Each data point represents the maximum Poisson input rate (provided by multiple 10 Hz sources) that the core can handle in real time.

6.4 Static synaptic processing performance

I profiled the performance of the new static synapse processors and found that their performance has improved over the current approach: down to 15 cycles to process a synapse. This saving is achieved because, as each synapse processor has to process only a single type of synapse, the synapse processing loop can be further optimised. Using this figure we can estimate the rate of incoming synaptic events that each synapse processor can handle.

$$\mu_{events} \approx \frac{200 \times 10^6}{15} \tag{6.1}$$

This suggests that each synapse processor can handle just over 13×10^6 synap-



(a) Current approach – 7 cores

(b) Synapse-centric approach – 3 cores

Figure 6.6: Distribution of 1024 neurons with static synapses each receiving 24 kHz input amongst SpiNNaker cores using current and synapse-centric approaches. Connection density is 20 %. Dashed outlines illustrate the processing that occurs on a single SpiNNaker core. Heights of synapse processing blocks are scaled to reflect synaptic event processing performance. Heights of neuron processing blocks are scaled to reflect number of neurons that can be simulated. Double arrows indicate communication on same core and thick grey arrows indicate communication through external memory buffers.

tic events per second which will be divided amongst the 1024 neurons whose synapses each static synapse processor can simulate. If it is then assumed that, based on the model of cortical connectivity described in section 2.2, each neuron receives 24 kHz of synaptic input then we can estimate that 1024 neurons' afferent synapses could be simulated using 2 synapse processors. To verify these results I repeated the benchmark described in section 6.1 on a population of 1024 neurons mapped to one neuron processor and one synapse processor using the new synapse-centric approach. Figure 6.5 shows that the peak performance of the synapse processor is, indeed, almost 13×10^6 synaptic events per second although this reduces significantly with sparser connectivity. However, because the number of postsynaptic neurons does not need to be reduced until all of the afferent synapses can be simulated on a single core and because the length of a row representing the same connectivity is 4× longer than it would be when using the current approach, this effect is significantly less pronounced. On this basis, just 2 synapse processors can handle 100 % connectivity and 3 can handle the same situation with 20 % connectivity. Therefore, including the neuron processor, 341 neurons can be simulated per core at 100 % connectivity and 256 per core at 20 % connectivity; a significant improvement over the 256 and 155 achieved using the current approach.

One potential downside of the synapse-centric approach is that transferring input via SDRAM from the synapse to the neuron processors every simulation time step requires extra external memory bandwidth. To determine whether this affects the scaling of the synapse-centric approach, I extended the benchmark to use multiple synapse processors with the inputs divided evenly amongst them. Figure 6.7 shows that, with up to 9 synapse processors, synaptic processing performance grows linearly, with each additional synapse processor adding approximately 10×10^6 synaptic events per second to the total performance. However, the

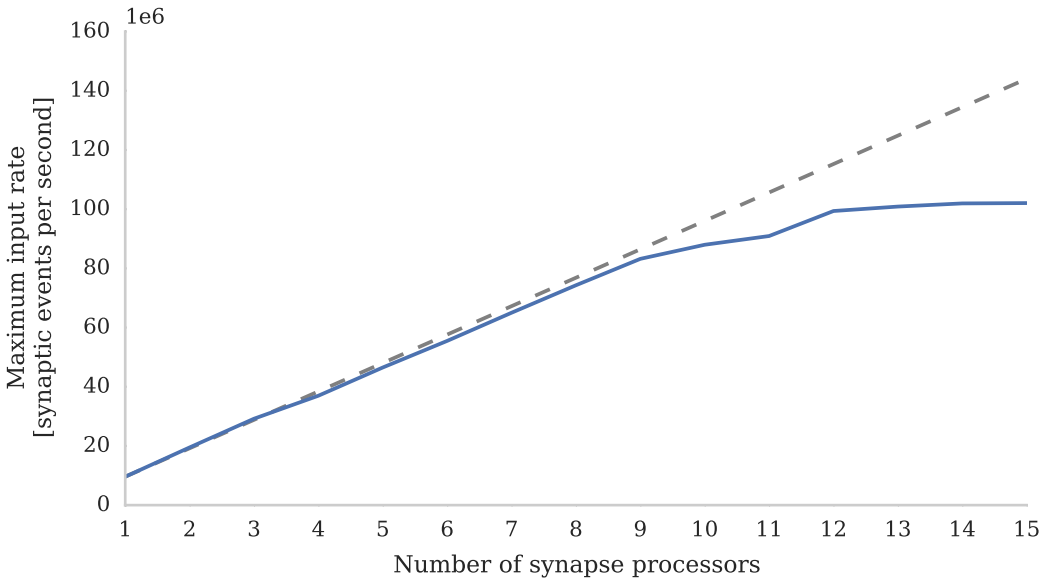


Figure 6.7: Performance of a SpiNNaker chip containing one neuron processor simulating population of 512 neurons and increasing numbers of synapse processors simulating the afferent static synapses associated with the population. Each data point represents the maximum Poisson input rate (provided by multiple 10 Hz sources) that the core can handle in real time. 20 % connection sparsity is used for all data points. The dashed line shows the linear scaling of the performance with one synapse processor.

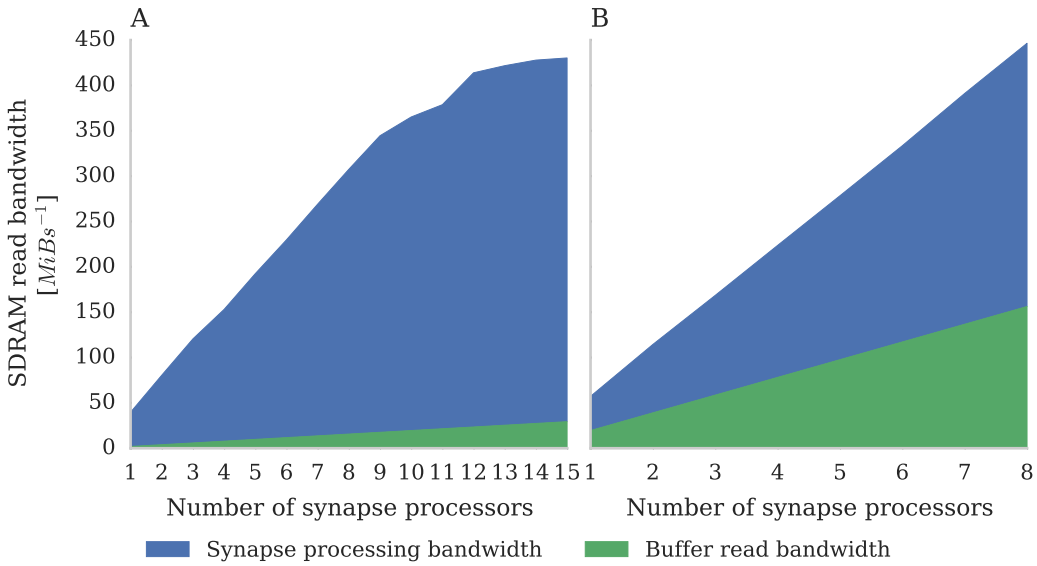


Figure 6.8: External memory read bandwidth used by a SpiNNaker chip containing increasing numbers of synapse processors simulating the afferent static synapses associated with 512 neurons. Colours indicate how much of this bandwidth is used transferring synaptic matrix rows and how much for transferring input currents to the neuron processor(s). **(A)** With a simulation time step of 1 ms where the 512 neurons are simulated on a single neuron processor. **(B)** With a simulation time step of 0.1 ms where the 512 neurons are simulated across 8 neuron processors.

performance plateaus with 12 synapse processors delivering a synaptic processing performance of around 100×10^6 synaptic events per second.

Fetching the synaptic matrix rows required by a single synapse processor requires approximately 40 MiB s^{-1} of external memory bandwidth and transferring the input currents associated with 512 neurons every 1 ms simulation time step requires approximately another 2 MiB s^{-1}. Figure 6.8a shows the external memory read bandwidth usage in the benchmark and – similarly to the performance shown in figure 6.7 – this increases linearly with up to 9 synapse processors and plateaus at 420 MiB s^{-1}.

If the simulation time step is reduced to 0.1 ms, the bandwidth required to transfer the input currents from each synapse processor increases to 20 MiB s^{-1}. Figure 6.8b shows the results of repeating the benchmark on a 0.1 ms simulation time step with 8 neuron processors and up to 8 synapse processors. Because of the increased bandwidth required to transfer input currents every 0.1 ms, this configuration has a significantly higher peak bandwidth of 450 MiB s^{-1}, but shows no sign of the performance plateauing.

To illustrate the advantages of this new simulator in the context of a more realistic network I ran several simulations of the network developed by Vogels and Abbott [110]. This network was designed as a medium for experimentation into signal propagation through cortical networks, but has subsequently been widely used as a benchmark [64]. The network consists of 10 000 integrate-and-fire neurons, split between an excitatory population of 8000 cells and an inhibitory population of 2000 cells. To be representative of long-range cortical connectivity these populations are randomly connected with a very low connection probability of 2 %. Table 6.1 shows that, if a 500 ms simulation of this network is run on SpiNNaker using either 1 ms or 0.1 ms time steps, the new approach requires fewer cores than the current approach. However, due to its small size and sparse

connectivity, each neuron in this network receives only 200 synaptic inputs; far below the degree of connectivity seen in the cortex and the performance limits of the synapse processors. Therefore I increased the connection density of the network to 10 % (the highest density at which Vogels and Abbott suggest their results hold) and increased the total number of neurons to 80 000 so that each neuron in the network receives 8000 inputs. Because the neurons in this network have both inhibitory and excitatory synapses, in the synapse-centric approach, they are simulated on separate synapse processors. Therefore an extra synapse processor – beyond the 3 previously calculated – is required to simulate the synapses associated with each 1024 neurons. As discussed in section 6.3 each neuron processor can simulate up to 1024 LIF neurons. However, processing this many neurons leaves insufficient time within a simulation time step to process the input from 4 synapse processors. Therefore I reduced the number of neurons simulated on each neuron processor to 512, resulting in an average of 170 neurons being simulated on each core. This is a significant improvement over the 60 neurons per core the benchmark – shown in figure 6.1 – suggests the standard approach can achieve at 10 % connectivity.

6.5 Plastic synaptic processing performance

I profiled the performance of a synapse processor core simulating synapses with pair-based STDP and an additive weight dependence [101]. Similarly to the static synapse processors, due to the optimisations made possible because only a single type of synapse is simulated on each synapse processor, the performance was somewhat improved over that presented in section 4.4. Based on the model obtained through this profiling we can estimate the rate of incoming synaptic events that each STDP synapse processor can handle using the following equation:

Number of neurons	Connectivity [%]	Simulator	Simulation time step [ms]	Number of cores			Neurons per core
				Neuron	Synapse	Total	
10000	2	Standard	1.0	40		40	250
		Synapse-centric	1.0	10	20	30	333
10000	2	Standard	0.1	157		157	64
		Synapse-centric	0.1	99	26	125	80
80000	10	Synapse-centric	1.0	157	314	471	170

Table 6.1: Simulations of the Vogels Abbott benchmark networks on SpiNNaker using synapse-centric and standard approaches.

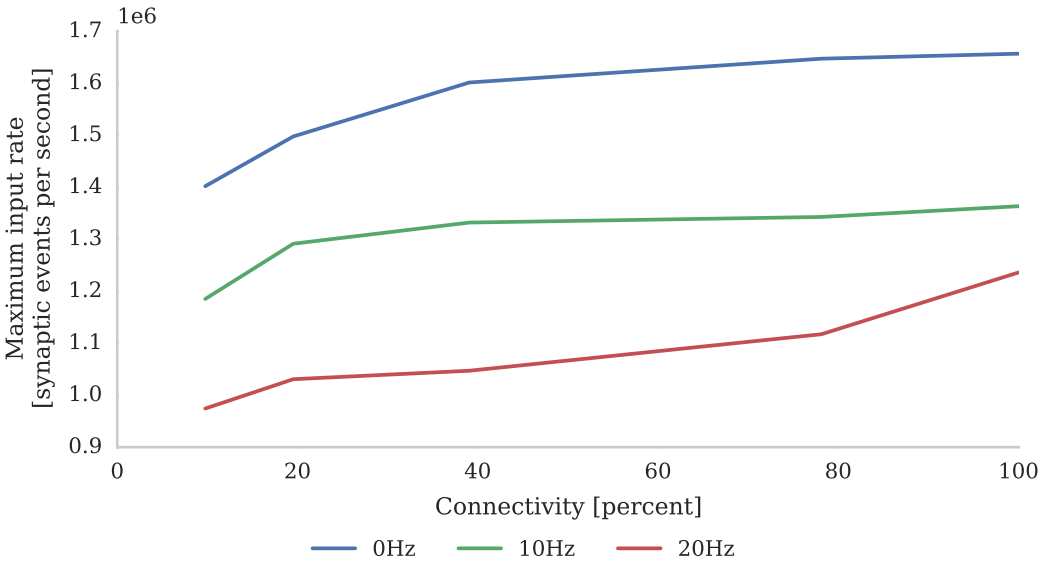


Figure 6.9: Performance of a synapse processor simulating the afferent STDP synapses associated with 512 neurons. Each data point represents the maximum Poisson input rate (provided by multiple 10 Hz sources) that the core can handle in real time.

$$\mu_{events} \approx \frac{200 \times 10^6}{107 + 30h} \tag{6.2}$$

Where h represents the average number of postsynaptic events in the history structure that require processing at each synapse. This models suggests that in the case where the pre and postsynaptic neurons are firing at approximately the same rate ($h = 1$) each synapse processor can handle just over 1.4×10^6 synaptic events every second. As discussed in section 6.3 the local memory requirements of the postsynaptic history structure mean that each STDP synapse processor can simulate the afferent synapses associated with 512 neurons. Therefore if we divide the total estimated performance between 512 neurons, and again use the model of cortical connectivity summarised in section 6.1, we can estimate that the afferent

synapses associated with the 512 neurons can be simulated using 9 synapse processors. To verify this performance I repeated the STDP benchmark described in section 6.1 using a population of 512 neurons mapped to one neuron processor and one synapse processor using the new synapse-centric approach. The results of this benchmark are presented in figure 6.9 and show that the peak performance is indeed nearly 1.4×10^6 synaptic events per second. Because processing an STDP synapse is significantly more costly than processing a static synapse, the fixed cost of processing a row is amortised over fewer synapses, meaning that 10 STDP synapse processors are sufficient to deliver the model of cortical connectivity down to just over 10 % connection sparsity. Therefore, taking into account the core used by the neuron processor, 46 neurons can be simulated per core, more than 4× the number possible when using the current approach with 20 % connectivity.

In chapter 5 I demonstrated how the spiking BCPNN learning rule [117] could be implemented efficiently on SpiNNaker, within the framework developed in chapter 4, and used to learn temporal sequences of neural activity within a modular attractor network. While this was the largest plastic neural network ever to be simulated on neuromorphic hardware, the training process was hampered by the inability of the approach described in section 4.3 to simulate neurons with different learning rules on their afferent synapses. This limitation meant that separate networks had to be simulated to train the AMPA and NMDA synapses, the learned weights downloaded, combined together and finally re-uploaded to the SpiNNaker machine for testing. This model also had several features that placed high demands on the local memory available to each core. Firstly BCPNN requires 32 bits of state to be stored with each event in the postsynaptic history structure rather than the 16 bits required by STDP synapses, meaning that a 10 entry postsynaptic history requires an extra 20 B of local memory for each neuron.

Additionally the model uses three synapse types (AMPA, NMDA and GABA) – each of which requires a separate input ring-buffer – and each neuron in the network also has several extra parameters used to configure a simple spike frequency adaptation mechanism [126]. These factors conspired to reduce the local memory available and, when combined with the high cost of simulating BCPNN synapses, meant that, although each neuron in the model only had 4000 inputs, only 75 neurons could be simulated on each core and the network could be run only at 0.5× real time.

Using the new approach I trained both the AMPA and NMDA plastic synapses of the network described in section 5.5 simultaneously on separate synapse processors, each with different BCPNN configurations. The implementation of BCPNN for use with the synapse-centric simulator and the modular attractor network are available from `https://github.com/project-rig/pynn_spinnaker_bcpnn`. Table 6.2 summarises the results of simulations of this modular attractor network with 4, 9 and 16 macrocolumns using the synapse-centric approaches. While, in all but the 4 macrocolumn configuration, these simulations require more cores than those presented in chapter 5 (see table 5.2) they allow the network to be simulated in real time in all configurations. The standard approach can only simulate the network in real time with 4 macrocolumns and would require the neural populations to be further sub-divided to achieve real time performance at larger scales. Furthermore, the comparison is somewhat unfair as, when using the standard approach, only one synapse type is learned at once meaning that, during the training phase when plasticity is enabled, each core needs to be capable of handling only half the rate of incoming synaptic events.

Num macrocolumns	Number of cores				
	Neuron	Synapse	Poisson	Stimulus	Total
4	12	44	8	4	68
9	27	180	27	18	252
16	48	448	48	32	576

Table 6.2: SpiNNaker simulations of the BCPNN modular attractor network at varying scales using synapse-centric approach.

6.6 Conclusions

In this chapter I have presented a new synapse-centric approach for mapping the simulation of highly-connected point neuron networks to SpiNNaker and have shown how this approach can significantly increase the size of network that can be simulated on a given SpiNNaker machine. In section 6.4 I analysed the peak synaptic processing performance of an entire SpiNNaker chip using this approach and found that, with up to 9 synapse processors running on the chip, performance scales linearly but plateaus with 12 synapse processors at around 100×10^6 synaptic events per second. This peak throughput requires $420\,\mathrm{MiB\,s^{-1}}$ of external memory read bandwidth which is significantly lower than the peak external memory read bandwidth of $600\,\mathrm{MiB\,s^{-1}}$ measured by Painkras *et al.* [84]. Therefore I believe that this plateau occurs when contention for access to the external memory increases the duration of each DMA transfer to the point where external memory latency can no longer be hidden. However, if the simulation time step is reduced to 0.1 ms – requiring input currents to be transferred from the synapse processors to the neuron processors 10× more frequently – $450\,\mathrm{MiB\,s^{-1}}$ of external memory read bandwidth can be obtained. This supports the view that the plateauing of performance is not due to the memory bandwidth being saturated. Furthermore, by simulating a more realistic network of 80 000 neurons each with 8000 sparsely

connected inputs, I demonstrate that 8 synapse processors and 4 neuron processors running on a SpiNNaker chip is likely to be a more typical configuration for simulating cortical networks with static synapses. This configuration is well within the region where figures 6.7 and 6.8 show linear performance scaling and leaves 4 cores free to provide additional background noise or stimuli to the neurons. In section 6.5 I analyse the performance of pair-based STDP synapses with an additive weight dependence and find that they are between 6.5× and 10× more costly to simulate than static synapses. This reduction in performance compared to static synapse processors corresponds to similar reductions in memory read bandwidth requirements meaning that the static network represents the worst case in terms of external memory bandwidth requirements.

In section 6.5 I demonstrated that this new approach offers significant efficiency savings when simulating cortical models with plastic synapses and also enables the simulation of neurons with multiple types of synapse. However, neurons in the cortex have many more degrees of heterogeneity particularly in the morphology and complexity of their dendritic trees [137]. In section 3.1.2 I briefly discussed the type of multi-compartmental models that can be used to simulate this heterogeneity and, potentially, the synapse-centric simulator could provide the basis for mapping such models onto SpiNNaker by adding *dendritic compartment processors*. The dendritic compartment processors would, like the current neuron processors, receive synaptic input from synapse processors through memory buffers. Additionally they would receive membrane voltages from neighbouring neuron and dendritic compartment processors through additional memory buffers. During each simulation time step the dendritic compartment processors would update the state of their dendritic compartment and write its membrane voltages to a memory buffer.

Chapter 7

Conclusions

Compared to older regions of the brain the neocortex has a relatively homogeneous structure. However, this structure is connected by highly plastic synapses whose efficacy and structure change dynamically in response to stimuli. While its dynamic nature allows the neocortex to perform a diverse range of functions, it also makes attempting to understand its structure through experiments on living biological tissue extremely difficult.

A promising alternative is, instead, to perform experiments on computer simulations of neocortical models. Unlike experiments on biological tissue such experiments allow all the input to the model be controlled and its outputs fully observed. However, due to both their plasticity and sheer number, efficiently simulating the synaptic connections of the neocortex is a difficult task.

One approach to this challenge is to a use a massively parallel neuromorphic architecture such as SpiNNaker which is designed specifically for simulating large networks of spiking neurons in real time. Unfortunately previous approaches developed for simulating synaptic plasticity on SpiNNaker have drastically impacted on the number of neurons a given SpiNNaker machine can simulate.

My first contribution, presented in chapter 4, is a new SpiNNaker synap-

tic plasticity implementation with lower algorithmic complexity than prior approaches and employing new low-level optimisations to better exploit the ARM instruction set. This new implementation almost doubles the performance of previous approaches and is now a key component of the SpiNNaker software developed for the Human Brain Project.

My second contribution, presented in chapter 5, is the first SpiNNaker implementation of the Bayesian Confidence Propagation Neural Network (BCPNN) learning rule. Bayesian inference provides an intuitive model of how our brains internalise uncertainty about the outside world and BCPNN can approximate this using spiking neurons. Using the BCPNN learning rule I built a simple neocortical model which was able to learn and replay sequences of neuronal activity – demonstrating a possible way in which temporal and spatial associations could be learnt in the neocortex.

Simulations of this neocortical model reveal however, that as we begin to simulate models with the degree of connectivity found in the neocortex, the current SpiNNaker simulator scales poorly. In fact, when simulating neurons with a biologically-plausible number of plastic synapses, the performance of a single SpiNNaker core can be 3× lower than the benchmarks presented in chapter 4 predict. Therefore, my final contribution, presented in chapter 6, is an entirely new "synapse-centric" SpiNNaker simulator which addresses this poor scaling and, I believe, is the most significant contribution of this thesis. In practice this new simulator *quadruples* the number of neurons with biologically plausible numbers of plastic synapses that can be simulated on a single SpiNNaker core when compared to the current SpiNNaker simulator. Using the new simulator a rerun of the simulations of the neocortical model performed in chapter 5 demonstrates that they can be performed in biological real time with two forms of simultaneously active synaptic plasticity.

Simulator	Synapse Type	Num neurons per core	Cores required	Card frames required
Current	Static	155	103 226	6
Synapse-centric	Static	256	62 500	4
Current	Plastic	10	1 600 000	87
Synapse-centric	Plastic	46	347 827	19

Table 7.1: Estimated SpiNNaker hardware requirements for simulation of mouse neocortex consisting of 1.6×10^7 neurons and 1.28×10^{11} synapses [30]. Connection sparsity is assumed to be 20 %. Plastic synapses are simulated using the implementation discussed in chapter 4.

Figure 7.1: 20 000 core SpiNNaker card frame.

Figure 7.2: 500 000 core 5 cabinet SpiNNaker system.

To put the improved performance of the new synapse-centric simulator in context, consider a full-size model of a mouse neocortex comprising 1.6×10^7 neurons and 1.28×10^{11} synapses [30]. Table 7.1 shows that, using the SpiNNaker simulator presented in chapter 3, a version of this model with static synapses could be simulated using 6 of the SpiNNaker card frames shown in figure 7.1. However, using the synapse-centric simulator, this could be reduced to only 4 such card frames – allowing the half million core machine shown in figure 7.2 to simulate two additional mouse neocortices. Furthermore, using the current SpiNNaker simulator and the synaptic plasticity implementation developed in chapter 4, a version of this model using plastic synapses would require 87 card frames – Almost double the 10 cabinets planned for the final SpiNNaker machine. However, using this same synaptic plasticity implementation with the new synapse-centric simulator, the plastic mouse neocortical simulation could be run on only 19 card frames – fitting easily on the system shown in figure 7.2.

7.1 SpiNNaker

In this thesis I have developed neural simulation tools optimised for the current SpiNNaker architecture. The synaptic plasticity implementation presented in chapter 4 exploits its ARM instruction set for improved performance and the synapse-centric simulator presented in chapter 6 parallelises the simulation of neurons and synapses amongst multiple cores of a SpiNNaker chip.

Because the current SpiNNaker architecture provides no other means of bulk on-chip communications, the cores used for the synapse-centric neural simulator communicate using memory buffers. In section 6.4 I demonstrated that the extra external memory bandwidth this requires is unlikely to saturate the memory bandwidth of the current SpiNNaker system.

Designs for a next-generation SpiNNaker system are already underway and, while its basic computational units are likely to be somewhat more powerful than those used by the current system, improved performance will largely be obtained by integrating more cores into each chip [138]. However, the gap in performance between DRAM and CPUs has increased since the SpiNNaker architecture was originally conceived, meaning that providing sufficient external memory bandwidth for a SpiNNaker chip with *more* cores is likely to present a significant challenge. These architectural pressures act to make bandwidth more precious. As discussed in section 6.6 the synapse-centric approach may also be a possible means of simulating multi-compartmental models of the type discussed in section 3.1.2 on SpiNNaker. Furthermore, to accurately simulate more complex models, smaller simulation time steps are likely to be necessary [82] which, as figure 6.8b showed, increase the frequency at which buffers have to be exchanged and further exacerbates the problem. These issues – highlighted in this work – have been addressed in the proposed design of the next-generation SpiNNaker system by employing a NoC architecture that allows cores direct access to the local memory of other cores.

Beyond its use by the synapse-centric simulator, the ability to share data amongst cores without sacrificing external memory bandwidth will allow applications to extract another level of (finer-grained) parallelism, which message passing alone cannot provide. This will have additional benefits for the system's fault tolerance as it can allow the contents of a crashed core's local memory to be transferred to another core allowing it to continue from the same state.

7.2 Models of the neocortex

The neocortical model presented in chapter 5 demonstrates a possible way that layers II/III of the neocortex may learn temporal and spatial associations. However, the sequences formed by these associations represent only one level of the hierarchy of sequences which Lashley [8] suggested is the basis of the complex behaviour performed by the mammalian neocortex. Exactly how models of this sort can be combined to facilitate the forward and backward flow of information discussed in section 2.2 remains unclear.

One possibility might be to replace the single inhibitory and excitatory populations which makes up each macrocolumn in the current model with a more realistic, layered, macrocolumn model such as that developed by Potjans and Diesmann [139]. Multiple macrocolumns, modelled in this way, could then be connected using plastic synapses to form the connectivity presented in section 2.2. However, this approach would drastically increase the complexity of the model and therefore the size of the parameter space required to configure it. Alternatively, both Bartlett and Sejnowski [140] and Johansson and Lansner [141] suggest that simpler multi-layer neocortical models can be built by adding a model of layer IV in which competitive learning occurs between the feedforward inputs to each macrocolumn.

A more fundamental limitation of the model presented in chapter 5 is that it uses an unrealistic learning paradigm. Learning is manually turned on while pertinent information is presented to the model and then turned off again afterwards. In reality, our brains are permanently awash with stimuli, but we remember only that which is novel, surprising or results in reward. Novelty, surprise and reward are believed to be communicated in the brain by neuromodulators such as dopamine, acetylcholine and noradrenaline (see Frémaux and Gerstner [142] for a review).

As discussed in section 5.6, neuromodulatory input can be incorporated into

the BCPNN learning rule using the κ parameter. However, as spikes can arrive at the synapse from neuromodulator-releasing populations at any time, the times of modulatory as well as postsynaptic spikes need to be integrated into the synaptic plasticity algorithm. Because entire populations of neuromodulator-releasing neurons can deliver modulatory input to a single synapse, the postsynaptic history structure presented in chapter 4 is not a viable means of storing them. Potjans *et al.* [143] extend the STDP algorithm developed by Morrison *et al.* [96] to support neuromodulated learning by introducing "volume transmitter" populations which handle all the incoming modulatory input to a virtual "volume" of neural tissue. These populations maintain a spike-history of all incoming modulatory spikes and deliver these to the synapses of neuronal populations within this volume, both at presynaptic spike times and after a fixed period so as to "flush out" the spike-history data structure and allow it to be kept relatively small. This approach has the potential to map well to the SpiNNaker architecture and could be used as the basis of a future SpiNNaker implementation of neuromodulated learning using BCPNN.

Bibliography

[1] B. Pakkenberg, D. Pelvig, L. Marner, M. J. Bundgaard, H. J. G. Gundersen, J. R. Nyengaard, and L. Regeur, "Aging and the human neocortex," *Experimental Gerontology*, vol. 38, no. 1, pp. 95–99, 2003.

[2] I. H. Stevenson and K. P. Kording, "How advances in neural recording affect data analysis.," *Nature Neuroscience*, vol. 14, no. 2, pp. 139–142, 2011, ISSN: 1097-6256. arXiv: NIHMS150003.

[3] S. W. Oh, J. A. Harris, L. Ng, *et al.*, "A mesoscale connectome of the mouse brain," *Nature*, vol. 508, no. 7495, pp. 207–214, 2014.

[4] D. C. Van Essen, S. M. Smith, D. M. Barch, T. E. Behrens, E. Yacoub, K. Ugurbil, W.-M. H. Consortium, *et al.*, "The WU-Minn human connectome project: An overview," *Neuroimage*, vol. 80, pp. 62–79, 2013.

[5] H. Markram, E. Muller, S. Ramaswamy, *et al.*, "Reconstruction and simulation of neocortical microcircuitry," *Cell*, vol. 163, no. 2, pp. 456–492, 2015, ISSN: 10974172.

[6] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[7] A. Lansner and A. Holst, "A higher order Bayesian neural network with spiking units," *International Journal of Neural Systems*, no. Pearl, pp. 1–16, 1996.

[8] K. Lashley, "The problem of serial order in behavior," *Cerebral Mechanisms in Behavior*, pp. 112–131, 1951.

[9] B. B. Averbeck, M. V. Chafee, D. A. Crowe, and A. P. Georgopoulos, "Parallel processing of serial movements in prefrontal cortex," *Proc Natl Acad Sci U S A*, vol. 99, no. 20, pp. 13 172–13 177, 2002, ISSN: 00278424.

[10] J. C. Knight, P. J. Tully, B. A. Kaplan, A. Lansner, and S. B. Furber, "Large-scale simulations of plastic neural networks on neuromorphic hardware.," *Frontiers in Neuroanatomy*, vol. 10, no. April, p. 37, 2016, ISSN: 1662-5129.

[11] J. Knight and S. Furber, "Synapse-centric mapping of cortical models to the SpiNNaker neuromorphic architecture," *Frontiers in Neuroscience*, vol. 10, p. 420, 2016, ISSN: 1662-453X.

[12] A. Mundy, J. Knight, T. C. Stewart, and S. Furber, "An efficient SpiNNaker implementation of the Neural Engineering Framework," in *The 2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015.

[13] C. Eliasmith and C. H. Anderson, *Neural Engineering*. MIT Press, 2003, ISBN: 978-0262550604.

[14] J. Knight, A. R. Voelker, A. Mundy, C. Eliasmith, and S. Furber, "Efficient SpiNNaker simulation of a heteroassociative memory using the neural engineering framework," in *The 2016 International Joint Conference on Neural Networks (IJCNN)*, 2016.

[15] D. MacNeil and C. Eliasmith, "Fine-tuning and the stability of recurrent neural networks.," *PloS ONE*, vol. 6, no. 9, e22885, 2011, ISSN: 1932-6203.

[16] A. R. Voelker, E. Crawford, and C. Eliasmith, "Learning large-scale heteroassociative memories in spiking neurons," *Unconventional Computation and Natural Computation (UCNC), The 13th International Conference on*, 2014.

[17] H. Markram, K. Meier, T. Lippert, *et al.*, "Introducing the human brain project," *Procedia Computer Science*, vol. 7, pp. 39–42, 2011.

[18] J. D. Zakis and B. J. Lithgow, "Neurone modelling using VHDL," in *The Inaugural Conference of the Victorian Chapter of the IEEE Engineering in Medicine and Biology Society*, Victoria, Australia, 1999.

[19] H. Dale, "Pharmacology and nerve-endings," *Journal of the Royal Society of Medicine*, vol. 28, no. 3, pp. 319–332, 1935.

[20] N. Spruston, "Pyramidal neurons: dendritic structure and synaptic integration.," *Nature Reviews. Neuroscience*, vol. 9, no. 3, pp. 206–221, 2008, ISSN: 1471-003X.

[21] T. Branco and M. Häusser, "The single dendritic branch as a fundamental functional unit in the nervous system," *Current Opinion in Neurobiology*, vol. 20, no. 4, pp. 494–502, 2010, ISSN: 09594388.

[22] B. W. Mel, "NMDA-based pattern discrimination in a modeled cortical neuron," *Neural Computation*, vol. 4, pp. 502–517, 1992, ISSN: 0899-7667.

[23] H. Ko, L. Cossell, C. Baragli, J. Antolik, C. Clopath, S. B. Hofer, and T. D. Mrsic-Flogel, "The emergence of functional microcircuits in visual cortex," *Nature*, vol. 496, no. 7443, pp. 96–100, 2013, ISSN: 0028-0836.

[24] P. Bach-Y-Rita, C. C. Collins, F. A. Saunders, B. White, and L. Scadden, "Vision substitution by tactile image projection," *Nature*, 1969.

[25] D. Hubel and T. N. Wiesel, "Functional architecture of macaque monkey visual cortex," in *Proc. R. SOC. Lond. B*, vol. 198, 1977, pp. 1–59.

[26] D. Y. Tsao, W. A. Freiwald, R. B. Tootell, and M. S. Livingstone, "A cortical region consisting entirely of face-selective cells," *Science*, vol. 311, no. 5761, pp. 670–674, 2006.

[27] V. B. Mountcastle, "The columnar organization of the neocortex," *Brain*, vol. 120, no. 4, pp. 701–722, 1997.

[28] D. P. Buxhoeveden and M. Casanova, "The minicolumn hypothesis in neuroscience," *Brain*, vol. 125, no. 5, pp. 935–951, 2002, ISSN: 14602156.

[29] C. Beaulieu and M. Colonnier, "Number and size of neurons and synapses in the motor cortex of cats raised in different environmental complexities," *Journal of Comparative Neurology*, vol. 289, no. 1, pp. 178–187, 1989.

[30] V. Braitenberg and A. Schüz, *Cortex: Statistics and geometry of neuronal connectivity*. Springer Science & Business Media, 2013.

[31] W. H. Bosking, Y. Zhang, B. Schofield, and D. Fitzpatrick, "Orientation selectivity and the arrangement of horizontal connections in tree shrew striate cortex," *The Journal of Neuroscience*, vol. 17, no. 6, pp. 2112–2127, 1997.

[32] P. S. Goldman and W. J. Nauta, "Columnar distribution of cortico-cortical fibers in the frontal association, limbic, and motor cortex of the developing rhesus monkey," in *Neuroanatomy*, Springer, 1993, pp. 561–581.

[33] J DeFelipe, M Conley, and E. Jones, "Long-range focal collateralization of axons arising from corticocortical cells in monkey sensory-motor cortex," *The Journal of Neuroscience*, vol. 6, no. 12, pp. 3749–3766, 1986.

[34] C. D. Gilbert and T. N. Wiesel, "Columnar specificity of intrinsic horizontal and corticocortical connections in cat visual cortex," *The Journal of Neuroscience*, vol. 9, no. 7, pp. 2432–2442, 1989.

[35] R. Perin, T. K. Berger, and H. Markram, "A synaptic organizing principle for cortical neuronal groups.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 108, no. 13, pp. 5419–5424, 2011, ISSN: 0027-8424.

[36] R. J. Douglas and K. A. Martin, "Neuronal circuits of the neocortex," *Annual Review of Neuroscience*, vol. 27, no. 1, pp. 419–451, 2004, ISSN: 0147-006X.

[37] S. R. y Cajal, *Histology of the nervous system of man and vertebrates*. Oxford University Press, USA, 1995, vol. 1.

[38] W. McCulloch and W Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[39] M. Minsky and S. Papert, *Perceptron: an introduction to computational geometry*. MIT press Boston, MA: 1969, vol. 19, p. 88, ISBN: 0-262-63022-2.

[40] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System modeling and optimization*, Springer, 1982, pp. 762–770.

[41] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[42] D. Tolhurst, "The amount of information transmitted about contrast by neurones in the cat's visual cortex," *Visual Neuroscience*, vol. 2, pp. 409–413, 1989.

[43] F. Rieke, *Spikes: Exploring the neural code*. MIT press, 1999.

[44] S. Panzeri, R. S. Petersen, S. R. Schultz, M. A. Lebedev, and M. E. Diamond, "Coding of stimulus location by spike timing in rat somatosensory cortex," *Neurocomputing*, vol. 44-46, pp. 573–578, 2002, ISSN: 09252312.

[45] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, pp. 500–544, 1952.

[46] M.-O. Gewaltig and M. Diesmann, "NEST (NEural Simulation Tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[47] B. W. Connors and M. J. Gutnick, "Intrinsic firing patterns of diverse neocortical neurons," *Trends in Neurosciences*, vol. 13, no. 3, pp. 99–104, 1990.

[48] C. M. Gray and D. A. McCormick, "Chattering cells: Superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex," *Science*, vol. 274, no. 5284, p. 109, 1996.

[49] J. R. Gibson, M. Beierlein, and B. W. Connors, "Two networks of electrically coupled inhibitory neurons in neocortex," *Nature*, vol. 402, no. 6757, pp. 75–79, 1999.

[50] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–72, 2003, ISSN: 1045-9227.

[51] M. J. Richardson, N. Brunel, and V. Hakim, "From subthreshold to firing-rate resonance," *Journal of Neurophysiology*, vol. 89, no. 5, pp. 2538–2554, 2003.

[52] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of Neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.

[53] A. Destexhe, M. Rudolph, and D. Paré, "The high-conductance state of neocortical neurons in vivo.," *Nature Reviews. Neuroscience*, vol. 4, no. 9, pp. 739–751, 2003, ISSN: 1471-003X.

[54] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 105, no. 9, pp. 3593–8, 2008, ISSN: 1091-6490. arXiv: 9906002 [cs].

[55] P. Dayan and L. F. Abbott, *Theoretical neuroscience*. Cambridge, MA: MIT Press, 2001, vol. 806.

[56] D. Attwell and A. Gibb, "Neuroenergetics and the kinetic design of excitatory synapses," *Nature Reviews. Neuroscience*, vol. 6, no. 11, pp. 841–849, 2005.

[57] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The art of scientific computing*. Cambridge University Press, 1997, vol. 2.

[58] S Rotter and M Diesmann, "Exact digital simulation of time-invariant linear systems with applications to neuronal modeling.," *Biological cybernetics*, vol. 81, no. 5-6, pp. 381–402, 1999, ISSN: 0340-1200.

[59] R. Brette, "Exact simulation of integrate-and-fire models with synaptic conductances," *Neural Computation*, vol. 18, no. 8, pp. 2004–2027, 2006.

[60] ——, "Exact simulation of integrate-and-fire models with exponential currents," *Neural Computation*, vol. 19, no. 10, pp. 2604–2609, 2007.

[61] A. Morrison, C. Mehring, T. Geisel, a. D. Aertsen, and M. Diesmann, "Advancing the boundaries of high-connectivity network simulation with distributed computing.," *Neural computation*, vol. 17, no. 8, pp. 1776–801, Aug. 2005.

[62] N. T. Carnevale and M. L. Hines, *The NEURON book*. Cambridge University Press, 2006.

[63] D. Goodman and R. Brette, "Brian: A simulator for spiking neural networks in Python.," *Frontiers in Neuroinformatics*, vol. 2, no. November, p. 5, 2008, ISSN: 1662-5196.

[64] R. Brette, M. Rudolph, T. Carnevale, *et al.*, "Simulation of networks of spiking neurons: a review of tools and strategies.," *Journal of computational neuroscience*, vol. 23, no. 3, pp. 349–98, 2007, ISSN: 0929-5313.

[65] RIKEN. (2013). Largest neuronal network simulation achieved using K computer, [Online]. Available: `http://www.riken.jp/en/pr/press/2013/20130802_1/`.

[66] H. Meuer. (2016). Top500 list - June 2016, [Online]. Available: `http://www.top500.org/list/2016/06/`.

[67] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural networks*, vol. 22, no. 5, pp. 791–800, 2009.

[68] A. K. Fidjeland and M. P. Shanahan, "Accelerated simulation of spiking neural networks using GPUs," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2010, pp. 1–8.

[69] E. Yavuz, J. Turner, and T. Nowotny, "GeNN: a code generation framework for accelerated brain simulations.," *Scientific reports*, vol. 6, no. November 2015, p. 18 854, 2016, ISSN: 2045-2322.

[70] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. October, pp. 1629–1636, 1990.

[71] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in Neuroscience*, vol. 9, no. April, pp. 1–17, 2015, ISSN: 1662-453X.

[72] B. V. Benjamin, P. Gao, E. McQuinn, *et al.*, "Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014, ISSN: 00189219.

[73] J. Schemmel, D Bruderle, A Grubl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, IEEE, 2010, pp. 1947–1950.

[74] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[75] Z. Or-Bach. (2014). FPGA as ASIC alternative: Past and future, [Online]. Available: `http://www.monolithic3d.com/blog/fpga-as-asic-alternative-past-and-future`.

[76] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[77] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, "Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot," in *Evolvable hardware, 2003. proceedings. nasa/dod conference on*, IEEE, 2003, pp. 189–198.

[78] S. W. Moore, P. J. Fox, S. J. Marsh, a. T. Markettos, and A. Mujumdar, "Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation," *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 133–140, 2012.

[79] M. Naylor, P. J. Fox, A. T. Markettos, and S. W. Moore, "Managing the FPGA memory wall: Custom computing or vector processing?" *2013 23rd International Conference on Field Programmable Logic and Applications, FPL 2013 - Proceedings*, 2013.

[80] A. P. Davison, D. Brüderle, J. Eppler, *et al.*, "PyNN: a common interface for neuronal network simulators.," *Frontiers in neuroinformatics*, vol. 2, no. January, p. 11, 2008, ISSN: 1662-5196.

[81] S. B. Furber, F Galluppi, S Temple, and L. A. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014, ISSN: 0018-9219.

[82] M. Hopkins and S. Furber, "Accuracy and efficiency in fixed-point neural ODE solvers," *Neural Computation*, vol. 27, pp. 2148–2182, 2015, ISSN: 1530888X.

[83] M. Moise, "A fixed point arithmetic library for SpiNNaker," PhD thesis, The University of Manchester, 2012.

[84] E. Painkras, L. A. Plana, J. Garside, *et al.*, "SpiNNaker: a 1-W 18-core system-on-chip for massively-parallel neural network simulation," *Solid-State Circuits, IEEE Journal of*, vol. 48, no. 8, pp. 1943–1953, 2013.

[85] E Nordlie, M. O. Gewaltig, and H. E. Plesser, "Towards reproducible descriptions of neuronal network models," *PLoS Computational Biology*, vol. 5, no. 8, e1000456, 2009, ISSN: 1553-7358.

[86] X Jin, S. Furber, and J. Woods, "Efficient modelling of spiking neural networks on a scalable chip multiprocessor," *The 2008 International Joint Conference on Neural Networks (IJCNN)*, pp. 2812–2819, 2008.

[87] G. Buzsáki and K. Mizuseki, "The log-dynamic brain: how skewed distributions affect network operations.," *Nature reviews. Neuroscience*, vol. 15, no. 4, pp. 264–78, 2014, ISSN: 1471-0048.

[88] G. Maimon and J. A. Assad, "Beyond poisson: increased spike-time regularity across primate parietal cortex," *Neuron*, vol. 62, no. 3, pp. 426–440, 2009, ISSN: 08966273.

[89] D. O. Hebb, *The organization of behavior*. Wiley & Sons, 1949.

[90] T. Bliss and T Lømo, "Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path," *The Journal of Physiology*, pp. 331–356, 1973.

[91] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex.," *The Journal of Neuroscience*, vol. 2, no. 1, pp. 32–48, 1982, ISSN: 0270-6474.

[92] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, pp. 267–273, 1983.

[93] P. U. Diehl and M. Cook, "Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware," in *The 2014 International Joint Conference on Neural Networks (IJCNN)*, 2014, pp. 4288–4295.

[94] W. Levy and O Steward, "Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus," *Neuroscience*, vol. 8, no. 4, 1983.

[95] G. Q. Bi and M. M. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type.," *The Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–72, 1998, ISSN: 0270-6474.

[96] A. Morrison, A. Aertsen, and M. Diesmann, "Spike-timing-dependent plasticity in balanced random networks.," *Neural computation*, vol. 19, no. 6, pp. 1437–67, 2007, ISSN: 0899-7667.

[97] R. Gütig, R Aharonov, S Rotter, and H. Sompolinsky, "Learning input correlations through nonlinear temporally asymmetric hebbian plasticity.," *The Journal of Neuroscience*, vol. 23, no. 9, pp. 3697–714, 2003, ISSN: 1529-2401.

[98] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, "Rate, timing, and cooperativity jointly determine cortical synaptic plasticity.," *Neuron*, vol. 32, no. 6, pp. 1149–64, 2001, ISSN: 0896-6273.

[99] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity.," *The Journal of Neuroscience*, vol. 26, no. 38, pp. 9673–82, 2006, ISSN: 1529-2401.

[100] A. Morrison, M. Diesmann, and W. Gerstner, "Phenomenological models of synaptic plasticity based on spike timing," *Biological Cybernetics*, vol. 98, pp. 459–478, 2008, ISSN: 03401200.

[101] S Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity.," *Nature neuroscience*, vol. 3, no. 9, pp. 919–26, 2000, ISSN: 1097-6256.

[102] T. P. Vogels, H. Sprekeler, F. Zenke, C. Clopath, and W. Gerstner, "Inhibitory plasticity balances excitation and inhibition in sensory pathways

and memory networks," *Science*, vol. 334, no. 6062, pp. 1569–1573, 2011, ISSN: 0036-8075.

[103] L. F. Abbott and S. B. Nelson, "Synaptic plasticity: taming the beast.," *Nature Neuroscience*, vol. 3 Suppl, pp. 1178–83, 2000, ISSN: 1097-6256.

[104] X. Jin, A. Rast, and F. Galluppi, "Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Ieee, 2010, pp. 1–8, ISBN: 978-1-4244-6916-1.

[105] H. Francis, "ARM DSP-enhanced extensions," 2001.

[106] A. H. Gittis, M. H. Setareh, and S. du Lac, "Mechanisms of sustained high firing rates in two classes of vestibular nucleus neurons: Differential contributions of resurgent Na, Kv3, and PPBK currents," *Journal of Neurophysiology*, pp. 1625–1634, 2010.

[107] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, "Connectivity reflects coding: A model of voltage-based STDP with homeostasis.," *Nature neuroscience*, vol. 13, no. December 2009, pp. 344–352, 2010, ISSN: 1097-6256.

[108] N. Brunel, "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons," *Journal of computational neuroscience*, vol. 8, no. 3, pp. 183–208, 2000, ISSN: 09252312.

[109] W. R. Softky and C. Koch, "The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs.," *The Journal of Neuroscience*, vol. 13, no. 1, pp. 334–50, 1993, ISSN: 0270-6474.

[110] T. P. Vogels and L. F Abbott, "Signal propagation and logic gating in networks of integrate-and-fire neurons," *The Journal of Neuroscience*, vol. 25, no. 46, pp. 10 786–10 795, 2005, ISSN: 0270-6474, 1529-2401.

[111] J. Rubin, D. Lee, and H. Sompolinsky, "Equilibrium properties of temporally asymmetric hebbian plasticity," *Physical Review Letters*, vol. 86, no. 2, pp. 364–367, 2001, ISSN: 0031-9007.

[112] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics.," *Neural Computation*, vol. 19, no. 11, pp. 2881–912, 2007, ISSN: 0899-7667.

[113] A. Saudargiene, B. Porr, and F. Wörgötter, "How the shape of pre-and postsynaptic signals can influence STDP: A biophysical model," *Neural Computation*, vol. 16, no. 3, pp. 595–625, 2004.

[114] A. Soltani and X.-J. Wang, "Synaptic computation underlying probabilistic inference.," *Nature neuroscience*, vol. 13, no. 1, pp. 112–9, 2010, ISSN: 1546-1726.

[115] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," *PLoS Computational Biology*, vol. 9, no. 4, 2013, ISSN: 1553734X.

[116] A. Lansner and Ö. Ekeberg, "A one-layer feedback artificial neural network with a Bayesian learning rule," *International journal of neural systems*, vol. 1, no. 01, pp. 77–87, 1989.

[117] P. J. Tully, M. H. Hennig, and A. Lansner, "Synaptic and nonsynaptic plasticity approximating probabilistic inference.," *Frontiers in synaptic neuroscience*, vol. 6, no. April, p. 8, 2014, ISSN: 1663-3563.

[118] a Sandberg, A Lansner, K. M. Petersson, and O Ekeberg, "A Bayesian attractor network with incremental learning.," *Network (Bristol, England)*, vol. 13, no. 2, pp. 179–94, 2002, ISSN: 0954-898X.

[119] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[120] J. Knight, *BCPNN spinnaker*, Apr. 2016.

[121] F. Fiebig and A. Lansner, "Memory consolidation from seconds to weeks: a three-stage neural network model with autonomous reinstatement dynamics," *Frontiers in Computational Neuroscience*, 2014.

[122] B. Vogginger, R. Schuffny, A. Lansner, L. Cederstrom, J. Partzsch, and S. Hoppner, "Reducing the computational footprint for real-time BCPNN learning," *Frontiers in Neuroscience*, vol. 9, no. January, pp. 1–16, 2015, ISSN: 1662-453X.

[123] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[124] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[125] M. Lundqvist, M. Rehn, M. Djurfeldt, and A. Lansner, "Attractor dynamics in a modular network model of neocortex," *Network: Computation in Neural Systems*, vol. 17, no. 3, pp. 253–276, 2006.

[126] Y. H. Liu and X. J. Wang, "Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron," *Journal of Computational Neuroscience*, vol. 10, no. 1, pp. 25–45, 2001, ISSN: 09295313.

[127] D. J. Amit, *Modeling brain function: The world of attractor neural networks*. Cambridge University Press, 1992.

[128] Cray, "Cray XC30-AC^{TM} supercomputer," Tech. Rep., 2013.

[129] M. Lundqvist, A. Compte, and A. Lansner, "Bistable, irregular firing and population oscillations in a modular attractor memory network," *PLoS Computational Biology*, vol. 6, no. 6, e1000803, 2010.

[130] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral Cortex*, vol. 17, no. 10, pp. 2443–2452, 2007, ISSN: 1047-3211.

[131] P. Berthet, J. Hellgren-Kotaleski, and A. Lansner, "Action selection performance of a reconfigurable basal ganglia inspired model with hebbian–bayesian go-nogo connectivity," *Frontiers in behavioral neuroscience*, vol. 6, 2012.

[132] A. Ponzi and J. Wickens, "Sequentially switching cell assemblies in random inhibitory networks of spiking neurons in the striatum," *The Journal of Neuroscience*, vol. 30, no. 17, pp. 5894–5911, 2010.

[133] X. Lagorce, E. Stromatias, F. Galluppi, L. a. Plana, S.-C. Liu, S. B. Furber, and R. B. Benosman, "Breaking the millisecond barrier on SpiNNaker: Implementing asynchronous event-based plastic models with microsecond resolution," *Frontiers in Neuroscience*, vol. 9, no. June, pp. 1–14, 2015, ISSN: 1662-453X.

[134] F. Galluppi, X. Lagorce, E. Stromatias, M. Pfeiffer, L. A. Plana, S. B. Furber, and R. B. Benosman, "A framework for plasticity implementation on the SpiNNaker neural architecture," *Frontiers in Neuroscience*, vol. 8, p. 429, 2015, ISSN: 1662-453X.

[135] W. Gerstner, R. Kempter, J. L. van Hemmen, and H. Wagner, "A neuronal learning rule for sub-millisecond temporal coding," *Nature*, vol. 383, no. LCN-ARTICLE-1996-002, pp. 76–78, 1996.

[136] A. Riehle, S. Grun, M. Diesmann, *et al.*, "Spike synchronization and rate modulation differentially involved in motor cortical function," *Science*, vol. 278, no. 5345, pp. 1950–1953, 1997, ISSN: 00368075.

[137] G. N. Elston, "Cortex, cognition and the cell: new insights into the pyramidal neuron and prefrontal function," *Cerebral Cortex*, vol. 13, no. 11, pp. 1124–1138, 2003, ISSN: 10473211.

[138] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," *ACM SIGOPS Operating Systems Review*, vol. 30, no. 5, pp. 2–11, 1996, ISSN: 01635980.

[139] T. C. Potjans and M. Diesmann, "The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model.," *Cerebral cortex (New York, N.Y. : 1991)*, 2012, ISSN: 1460-2199.

[140] M. Bartlett and T. Sejnowski, "Learning viewpoint-invariant face representations from visual experience in an attractor network," *Network: Computation in Neural Systems*, vol. 9, no. 3, pp. 399–417, 1998, ISSN: 0954-898X.

[141] C. Johansson and A. Lansner, "Towards cortex sized artificial neural systems.," *Neural networks : The official journal of the International Neural Network Society*, vol. 20, no. 1, pp. 48–61, 2007, ISSN: 0893-6080.

[142] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity and theory of three-factor learning rules," *Frontiers in Neural Circuits*, vol. 9, no. 85, p. 85, 2016, ISSN: 1662-5110.

[143] W. Potjans, A. Morrison, and M. Diesmann, "Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity.," *Frontiers in computational neuroscience*, vol. 4, no. November, p. 141, 2010, ISSN: 1662-5188.