SCALABILITY AND ROBUSTNESS OF ARTIFICIAL NEURAL NETWORKS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2016

By
Evangelos Stromatias
School of Computer Science

Contents

Word Count: 37448

List of Tables

List of Figures

Abstract

Artificial Neural Networks (ANNs) appear increasingly and routinely to gain popularity today, as they are being used in several diverse research fields and many different contexts, which may range from biological simulations and experiments on artificial neuronal models to machine learning models intended for industrial and engineering applications. One example is the recent success of Deep Learning architectures (e.g., Deep Belief Networks [DBN]), which appear in the spotlight of machine learning research, as they are capable of delivering state-of-the-art results in many domains.

While the performance of such ANN architectures is greatly affected by their scale, their capacity for scalability both for training and during execution is limited by the increased power consumption and communication overheads, implicitly posing a limiting factor on their real-time performance. The on-going work on the design and construction of spike-based neuromorphic platforms offers an alternative for running large-scale neural networks, such as DBNs, with significantly lower power consumption and lower latencies, but has to overcome the hardware limitations and model specialisations imposed by these type of circuits. SpiNNaker is a novel massively parallel fully programmable and scalable architecture designed to enable real-time spiking neural network (SNN) simulations. These properties render SpiNNaker quite an attractive neuromorphic exploration platform for running large-scale ANNs, however, it is necessary to investigate thoroughly both its power requirements as well as its communication latencies.

This research focusses on around two main aspects. First, it aims at characterising the power requirements and communication latencies of the SpiNNaker platform while running large-scale SNN simulations. The results of this investigation lead to the derivation of a power estimation model for the SpiNNaker system, a reduction of the overall power requirements and the characterisation of the intra- and inter-chip spike latencies. Then it focuses on a full characterisation of spiking DBNs, by developing a set of case studies in order to determine the impact of (*a*) the hardware bit precision; (*b*) the input noise; (*c*) weight variation; and (*d*) combinations of these on the classification performance of spiking DBNs for the problem of handwritten digit recognition. The results demonstrate that spiking DBNs can be realised on limited precision hardware platforms without drastic performance loss, and thus offer an excellent compromise between accuracy and low-power, low-latency execution. These studies intend to provide important guidelines for informing current and future efforts around developing custom large-scale digital and mixed-signal spiking neural network platforms.

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

I would like to express my gratitude to my supervisor Professor Steve B. Furber and to the Engineering and Physical Sciences Research Council (EPSRC) for funding this research [EP/G015740/1]. In addition, I would like to thank my MSc dissertation supervisor Dr. John Marsland; thanks to him I had the opportunity to learn and experiment with spiking neural networks for the first time. I would also like to thank my BSc final year project supervisor Dr. Grigoris Nikolaou who taught me how to approach and solve research problems and encouraged me to investigate challenges and ideas further.

I would also like to thank all of my collaborators for their invaluable feedback, ideas and papers we have written together. Special thanks to Francesco Galluppi, Cameron Patterson, Shih-Chii Liu, Michael Pfeiffer, Danniel Neil, Xavier Lagorce, Ryad Benosman, Luis Plana, Qian Liu, Garibaldi Pineda Garcia and Maria Teresa Serrano Gotarredona. I would also like to thank the organisers of the neuromorphic workshops that took place in Capocaccia and Telluride as they had an important impact on the research presented in this thesis. Many early implementations and experiments described here were developed during these workshops.

Additionally, I would like to thank my friends and colleagues in Manchester: Yaman Cakmakci, Paris Yiapanis, Ilia Pietri, Patrick Camilleri, Kostas Seichidis, Tom Sharp, Martin Grymel, Geoffrey Ndu, Matt Orlinski, Colin Barrett, Mustafa I-cant-remember-the-last-name, Ioanna Alifieraki, Unai Lopez, Evie Andrew, James Clarkson, Bernard D'Antras, Mona Demaidi, Cosmin Gorgovan, Nicolae Bogdan Grigore, Jonathan Heathcote, Mohsen Ghasempour, Farideh Jalalinajafabadi, Mahdi Jelodari, James Knight (too many Ichnusas in Capocaccia), Andrew Leeming, Gengting Liu, Andrew Mundy, Mireya Paredes Lopez, Andrey Rodchenko, Athanasios Stratikopoulos, Sebastian Werner, Guangda Zhang, Wei Song. The academic stuff in the APT group: Jim Garside, Dirk Koch, David Lester, Mikel Lujan , Javier Navaridas, Eva M. Navarro-Lopez, Vasilis Pavlidis, Antoniu Pop, John V. Woods (many thanks for all the help with

the thesis), Simon Davidson, Andi Drebes, Michael Hopkins, Abbas Kiasari, Christos Kotselidis, John Mawer, Joanna Moy, Andy Nisbet, Alex Rast, Andrew Rowley, Steve Temple (best sarcasm award :)), Will Toms, Dave Clark, Steve Rhodes, and Jeff Pepper. Finally and most importantly Nikos Rizos for the extra support during my last months in Manchester.

I would also like to thank some researchers and friends from the neuromorphic community for all the fruitful conversations and help during the various conferences and workshops. Many thanks to Zaf Fountas, Filipe Peliz Pinto Teixeira, Tobi Delbruck, Peter Diehl, Giacomo Indiveri, Saray Soldado, Timos Moraitis, Gaby Michel, Daniel Rasmussen, Chris Eliasmith, Terry Stewart, Nicolai Waniek, Cristian Axenie, Christian Denk, Jorg Conradt, Luis Camunas, Bernabe Linares-Barranco, Alejandro Linares-Barranco, Maria Teresa Serrano Gotarredona, Garrick Orchard, Greg Cohen, Andr van Schaik, Jonathan Tapson, Tara Hamilton, Jens Burger, James O'Sullivan, Mick Crosse, Anahita Mehta, Dimitra Emmanouilidou, Aleksandrs Ecins, Tomas Figliolia, Alejandro Pasciaroni.

Many thanks to the staff of Starbucks located next to the APT lab (Kayliegh, Laurita and Magda) for providing the necessary amount of caffeine to keep up with this research.

I am also grateful for my family and friends in Greece and around the world. Extra thanks to Marios Daoutis for his help reviewing this thesis. Finally, special thanks to Lenka Vysohlidova for the final push and the extra inspiration and motivation.

"The most beautiful people we have known are those who have known defeat, known suffering, known struggle, known loss, and have found their way out of the depths. These persons have an appreciation, a sensitivity, and an understanding of life that fills them with compassion, gentleness, and a deep loving concern.
Beautiful people do not just happen."
- Elisabeth Kübler-Ross
(July 8, 1926 - August 24, 2004)

Chapter 1

Introduction

Over many decades researchers from diverse scientific areas have used simulated neural networks in their experimentation. For computational neuroscientists, the focus is to create and test model hypotheses based on results retrieved from in-vivo or in-vitro experimentation. Large-scale simulations [Markram, 2006; Eliasmith et al., 2012] of neural tissue offer an attractive alternative methodology for investigating the functionality of different brain regions as it allows greater observability, experimental control and reproducibility. Computer scientists, on the other hand, inspired by the brain's inherent massive parallelism, energy-efficiency and tolerance to defects, seek to explore novel computational paradigms by exploiting the computational capabilities of networks of neurons for accelerating efficiently cognitive tasks [Aleksander, 1990; Maass and Markram, 2004; Furber and Temple, 2007].

In 1943 Warren McCulloch and Walter Pitts [McCulloch and Pitts, 1943] introduced the first artificial neuron model known as the Threshold Logic Unit (TLU). The TLU was a basic unit that summed its weighted inputs and produced a binary output if the sum exceeded a threshold. This model was used by researchers to investigate biological processes in the brain, as well as, a new computational framework in the field of artificial intelligence. Later in that decade, Donald Hebb in his book entitled *The organisation of behaviour* [Hebb, 1949] described his theory on neural plasticity known as Hebbian learning, which is often paraphrased as *"Neurons that fire together wire together"*. His theory was not only significant to the field of neuropsychology but also was later used as an unsupervised learning method in Artificial Neural Networks (ANNs). Frank Rosenblatt in the late 1950s introduced a two-layer network composed of TLUs, known as the perceptron [Rosenblatt, 1958], which was capable of classifying linearly separable patterns. Interest in ANNs stagnated due to a publication from Minsky

and Papert [1969] where they proved that two-layer neural networks were incapable of solving nonlinear separable problems, such as the exclusive-or function. Gradually the binary activation function was replaced with a nonlinear continuous function to model the firing rates of biological neurons in the brain, and more layers were added. The interest in ANNs revived with the development of the backpropagation algorithm by Werbos [1974] that enabled Multi-Layered Perceptrons (MLPs), with a nonlinear continuous activation functions, to be trained efficiently and in a supervised manner. ANNs stagnated once again in the early 1990s when Boser et al. [1992] introduced the Support Vector Machines (SVMs) as a way to create nonlinear classifiers. SVMs had better theoretical properties and outperformed ANNs in many tasks for several years.

While ANNs are once again a very hot topic in the field of machine learning with the advent of deep architectures [Hof, 2013], Spiking Neural Networks (SNNs), also referred to as the third generation of ANNs [Maass, 1997], were introduced in order to increase the level of biological realism in the neural network simulations. The main difference between SNNs and ANNs is the concept of time. Typically in ANNs, all neurons generate an output synchronously at each propagation cycle (MLP), which can be seen as the normalised firing rate of a neuron within a period of time. In contrast, each spiking neuron has a membrane potential which varies with time and input signals. When the membrane potential reaches a threshold value a spiking neuron generates a stereotypical event, also known as spike or action potential (AP), which travels along the axon to the synapses and alters the membrane potential of the target neurons. The first detailed model of a spiking neuron was described by Hodgkin and Huxley [1952], based on the experiments they performed on the giant axon of the squid. Since then networks of spiking neurons have been used to test and validate hypothesis of the functionality of neuronal circuits [Eliasmith et al., 2012], and as a new computational framework [Maass and Markram, 2004]. The interest in modelling the brain in greater detail is reflected by the scientific community with efforts like the *Human Brain Project* (HBP) [HBP, 2013] and the *Brain Research through Advancing Innovative Neurotechnologies*[1] (BRAIN) initiative. In addition, inspired by the parallel nature and efficiency of the human brain and from the need to model analogies of the human brain in computers, a research track has been spawned that investigates the idea of simulating neurons and synapses directly on hardware. This is reflected by the industry with projects like *SyNAPSE* [Sawada and Modha, 2013] from DARPA/IBM, aimed for executing cognitive tasks in an energy efficient manner and in real-time and *"to bring the sort of intelligence that*

[1] `http://www.braininitiative.nih.gov`

people usually associate with the cloud down to the handset" [Monroe, 2014]. In the past several research groups have investigated the idea of simulating MLPs directly on hardware in order to accelerate a number of applications efficiently and as a method to address potential future power limitations of conventional computer architectures known as the *dark silicon*[2] effect [Esmaeilzadeh et al., 2012a]. Results revealed that hardware accelerators based on ANNs are defect-tolerant [Temam, 2012] and have shown speed-ups and energy savings [Esmaeilzadeh et al., 2012b] when compared to conventional computers.

Spiking neural networks can be simulated with different levels of abstraction and granularity. Single compartment models [Moratal, 2012] are neuron models that capture the fundamental dynamics of biological neurons and due to their low computational cost are suitable for large-scale simulations [Izhikevich, 2004]. They are also particularly suited to biological real-time simulations, as this permits larger-scale neural networks to be created, whilst minimising power consumption; for instance to run biological models embodied in robots [Galluppi et al., 2012a] or use a retina to model the response of the visual system [Galluppi et al., 2012c]. In the past, large-scale simulations of spiking neural networks have been successfully executed on general-purpose supercomputers [Markram, 2006; Izhikevich and Edelman, 2008; Ananthanarayanan et al., 2009]. However, while supercomputers offer significant parallelism and great opportunity for model flexibility, they suffer from large electrical power demands, which are rarely reported, and from communication bottlenecks when simulating spiking neural networks. Wong et al. [2013] simulated 53×10^{10} neurons with 1.37×10^{14} synapses on a Sequioa - BlueGene supercomputer. The simulation ran $1542\times$ more slowly than biological real-time and the largest cost reported was communicating the spikes via MPI messaging. Power dissipation was omitted, but the TOP500 [top] supercomputer list states that the peak power dissipated by the Sequia - BlueGene/Q is 7,890 kW. There are also certain cases where real-time performance of a neural simulation is a desirable feature. Once such example is when cognitive neuroscientists and roboticists would like to test and validate their hypotheses using embodied agents [Galluppi et al., 2012c,a] interacting with their environment or by interfacing with biologically inspired sensors [Lichtsteiner et al.,

[2]Dark silicon is a term used to describe the amount of silicon that cannot be powered-on at the nominal operating voltage for a given thermal design power (TDP) constraint. A study by Esmaeilzadeh et al. [2012a], which employed two different scaling models for future technology nodes from 45 nm to 8 nm and two different classes of multi-core CPUs, GPUs and hybrid topologies, while keeping the power and area budget fixed, revealed that regardless of the chip architecture and topology, multi-core scaling is power limited. At the 22 nm technology node 21% of the transistors would have to be powered off, while at the 8 nm node the number of the switched off transistors increases to 50%.

2008; Leñero-Bardallo et al., 2011; Liu et al., 2010].

Neuromorphic engineering, a term coined by Mead [1990], is an interdisciplinary field which takes inspiration from biology, physics, computer science and engineering to design hardware models of neural and sensory systems. Neuromorphic engineering originally aimed at exploiting sub-threshold transistor dynamics to emulate neurons in silicon, efficiently and in real-time. Today, this term has been expanded to include mixed analogue/digital very large scale integration (VLSI) circuits, digital hardware implementations and also biologically inspired sensory processing systems [Lichtsteiner et al., 2008; Leñero-Bardallo et al., 2011; Liu et al., 2010]. By simulating neurons directly on hardware it is possible to overcome the synchronisation and communication overheads of conventional computers. Recently, TrueNorth [Merolla et al., 2014b] simulated a million spiking neurons in real-time while consuming 63 mW; the identical network executed on an optimised software simulator Compass [Preissl et al., 2012] was 100 to 200$\times$ slower than real-time and consumed 100,000 to 300,000$\times$ more energy per synaptic event [Merolla et al., 2014b]. The neuromorphic approach can be very power efficient, as neuron dynamics are implemented directly in silicon but many neuromorphic systems are highly optimised to a particular neural model and offer minimal configurable interconnectivity, often limited by wiring density. Most large-scale systems have overcome the latter by employing alternative communication approaches including using an Address Event Representation (AER) packet-based infrastructure, where only the address of the neuron that fired or the destination is transmitted, to enable connectivity and propagate spikes. To overcome the expense and effort of producing a custom chip, some research groups have focused their research on more off-the-shelf configurable systems. Whilst Graphical processing units (GPUs) and field programmable gate arrays (FPGAs) are excellent platforms for parallel computation their memory access bandwidth is a bottleneck. For very large-scale real-time simulations of SNNs on general programmable platforms it is typically not the computational cost, but the system communications that is the prime limiting factor [Brette and Goodman, 2012; Moore et al., 2012].

1.1 Background

Interest in ANNs got renewed over the recent years mainly due to the development of Deep Learning architectures, which are inspired by advances in neuroscience and are loosely based on how information is processed in the human brain. Currently,

deep neural networks represent the state-of-the-art solution for virtually all relevant machine learning, computer vision, and speech recognition benchmarks [Schmidhuber, 2015; Hof, 2013]. Their advantage over shallow architectures lies in their ability to extract hierarchies of increasingly abstract relevant features, which give rise to a data representation that lends itself for task-specific optimisation. Whereas convolutional networks [LeCun et al., 1998b; Sermanet et al., 2013] currently outperform other architectures on many vision tasks, the alternative architecture of Deep Belief Networks (DBNs) [Hinton and Salakhutdinov, 2006] remains very popular due to its ability to learn from large unlabelled datasets [Le et al., 2012], and because of its dual role as classifier and generative model of the data. In addition, DBNs have been shown to improve theoretical performance bounds by adding additional layers of neurons [Hinton and Salakhutdinov, 2006]. Although training larger and larger networks is currently the focus of academic and industrial research, this has led to growing demands on hardware platforms for deep learning. While training remains the biggest bottleneck, and some of the biggest networks trained to date have required days or weeks on high-performance computing infrastructure [Dean et al., 2012; Le et al., 2012], the sheer size of the resulting network calls for special purpose or GPU hardware acceleration to make the system run close to real-time [Farabet et al., 2011]. However, low-latency and real-time execution are key demands for mobile and robotic systems, which have limited computing resources and power but require quick system responses. A recently proposed solution to overcome the energy demands, communication overhead, and high response latencies of DBNs is to transform them into spiking neural networks, thereby exploiting the energy efficiency of event-based updates and communication [O'Connor et al., 2013]. Furthermore, the proposed framework, which has shown the desired low latency and high efficiency is targeted for implementation on event-based neuromorphic hardware platforms. Event-driven networks can have higher energy efficiency because a clock is not used in the network simulation, and not every neuron updates in every time step. The efficiency of the event-driven platform TrueNorth [Merolla et al., 2014b] is around 46 GSops/W, where Sops stands for *synaptic operations per second*. Implementing spiking DBNs on neuromorphic hardware will potentially enable them to perform in a low-latency and energy-efficient manner paving the way for neuromorphic hardware accelerators suitable for accelerating machine learning tasks. However, as neuromorphic platforms come in a various forms [Liu et al., 2015] it is important to investigate how their performance will be affected by hardware constraints imposed by different hardware implementations and input sensor noise.

The Spiking Neural Network Architecture, or SpiNNaker, is an application-specific integrated circuit (ASIC) designed by the Advanced Processor Technologies (APT) group, at the University of Manchester, to enable the energy-efficient and scalable simulation of SNNs [Furber and Brown, 2009; Furber et al., 2014]. Each SpiNNaker chip uses low-power, programmable embedded-type processors in conjunction with an efficient novel interconnection fabric. By connecting together a great number of SpiNNaker chips, a SpiNNaker machine is formed that is capable of providing support for very large networks of flexibly modelled neurons and synapses. The general programmability of SpiNNaker's processors allows experimental investigation of customised neural, synapse models, new plasticity rules [Galluppi et al., 2014b] and the utilisation of neuromorphic sensors [Galluppi et al., 2012c]. Models with diverse detail and precision are therefore supported, even heterogeneously within the same simulation. This flexibility positions SpiNNaker as an excellent exploration platform for the very active neuroscience and neuromorphic computing research area. In this context SpiNNaker may be used as a tool for investigating the performance and real-time requirements of spiking DBNs. Finally, it is estimated that a single SpiNNaker chip will dissipate less than 1 W during neural simulations [Furber and Brown, 2009]; however, as the size of the SpiNNaker machines scale up so does the total energy. A detailed power characterisation of the SpiNNaker chips will allow to estimate the requirements of the larger systems and also provide the necessary information for further optimisations.

1.2 Motivation and Research Aims

Benchmarking power figures for neurally-inspired hardware is challenging due to the specificity of different architectures and of models simulated on them. The research presented in this thesis aims to investigate the power requirements of the SpiNNaker platform while running large-scale spiking neural network simulations. Areas of particular interest are the power dissipated by the SpiNNaker chips at different stages of execution, the energy required to simulate a neuron per millisecond and per synaptic event. The outcome of this investigation will lead to the derivation of a power estimation model for the SpiNNaker system. This research also aims to characterise the intra and inter-SpiNNaker chip spike latencies imposed by the fabric and the software overheads, and to optimise the overall energy usage.

Spiking DBNs offer an attractive approach to neuromorphic accelerators due to their scalability, low-latency and potential energy-efficiency as recently demonstrated

by a software implementation [O'Connor et al., 2013]. However, gains in efficiency should not be outweighed by losses in classification performance due to computation with spikes instead of real numbers, or due to limitations of the hardware compared to conventional computers. The hope is that in networks of such large size, numerical imprecision would rather cancel out than accumulate. In the conversion of [O'Connor et al., 2013] from digital to spiking DBNs in software only small performance losses were observed and overall very good performance was reached.

This thesis aims to perform a full characterisation of spiking DBNs by utilising a reference software spiking neural network simulator [Goodman and Brette, 2008], and by developing a set of case studies to determine the impact of the hardware bit precision, the input noise, weight variance, and combinations on the classification performance of a deep network for handwritten digit recognition [LeCun et al., 1998a]. These studies will provide important guidelines for informing current and future efforts to develop custom large-scale digital and mixed-signal spiking network platforms such as SpiNNaker, TrueNorth, Neurogrid, Bluehive and BrainScales [Merolla et al., 2014b,a; Benjamin et al., 2014; Pfeil et al., 2013, 2012; Moore et al., 2012], as well as hardware learning circuits that can train synaptic weights in DBNs [Mitra et al., 2009; Neftci et al., 2014]. The outcome of this research aims to demonstrate that spiking DBNs can be realised on limited precision hardware platforms without drastic performance loss, and thus offer an excellent compromise between accuracy and low-power, low-latency execution. SpiNNaker will be used as an exploration platform to verify the correctness of the results, classification latencies of the software simulator and to estimate the scalability, in terms of power requirements, of the SpiNNaker platform.

1.3 Contributions and Publications

The main contributions of this Thesis include:

- A method to characterise the power dissipation of a SpiNNaker machine based on the number of neurons, synapses and the activity of a neural network that lead to a derivation of a power estimation equation [Stromatias et al., 2013], and the implementation of a novel suspend mode for reducing the overall energy usage [Stromatias et al., 2014]. In addition, it demonstrated the largest real-time recurrent spiking neural network simulation at the time of publication [Stromatias et al., 2013]. This is covered in Chapter 4.

- A characterisation of the spike latencies of the SpiNNaker platform as imposed by the communications fabric and the software overheads. This is the author's contribution to a work that has been published in the Journal of Frontiers in Neuromorphic Engineering [Lagorce et al., 2015] and which investigated the regimes where microsecond operation on SpiNNaker is possible. More details regarding this are presented in Chapter 4.

- An investigation on how hardware constraints impact the performance of spiking deep belief networks implemented on neuromorphic hardware. In particular how the limited bit precision during execution, the impact of silicon mismatch in the synapses and the noise in the spiking input signal impact the performance of these neural networks. Results of this contribution are covered in more detail in Chapter 5 and have been published in the Journal of Frontiers in Neuromorphic Engineering [Stromatias et al., 2015c].

- An implementation of a real-time, energy-efficient handwritten digit classification system on SpiNNaker by utilising spiking DBNs, and a comparison with previously published hardware implementations. In addition, an estimation of the power dissipation of larger spiking DBNs running on SpiNNaker machines. Results of this contribution have been accepted in Stromatias et al. [2015a,b], and are covered in Chapter 5.

1.3.1 Journal Papers

- F. Galluppi, X. Lagorce, E. Stromatias, M. Pfeiffer, L. Plana, S. B. Furber, and R. Benosman, *A framework for plasticity implementation on the SpiNNaker neural architecture*, Journal of Frontiers in Neuromorphic Engineering, 2014. Contributed in designing experiments for testing the BCM learning rule and analysing results.

- X. Lagorce, E. Stromatias, F. Galluppi, L. A. Plana, S-C Liu, S. B. Furber, R. Benosman, *Breaking The Millisecond Barrier On SpiNNaker: Asynchronous Event-Based Models With Microsecond Resolution And Plasticity*, Journal of Frontiers in Neuromorphic Engineering, 2015. Contributed in characterising the latencies of the SpiNNaker platform as imposed by the communication fabric and the software overheads. Results from this Journal paper can be found in Chapter 4.

- E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber and S-C. Liu, *Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms*, Journal of Frontiers in Neuromorphic Engineering, 2015. Results from this Journal can be found in Chapter 5.

- Q. Liu, G. Pineda-Garcia, E. Stromatias, T. Serrano-Gotarredona, and S. B. Furber, *Benchmarking Spike-Based Visual Recognition: a Dataset and Evaluation*, submitted to the Journal of Frontiers in Neuromorphic Engineering, 2015. Contributed by providing a review of the spike-based Deep Belief Networks and SpiNNaker implementations, as well as, a brief comparison of neuromorphic platforms in terms of their energy usage and precision.

1.3.2 Conference Papers

- E. Stromatias, F. Galluppi, C. Patterson, S. Furber, *Power Analysis of Large-Scale, Real-Time Neural Networks on SpiNNaker*, in proceedings of International Joint Conference on Neural Networks (IJCNN), Dallas, Texas, USA, August 4-9, 2013. Results from this paper can be found in Chapter 4.

- E. Stromatias, C. Patterson, S. Furber, *Optimising the Overall Power Usage on the SpiNNaker Neuromimetic Platform*, in proceedings of International Joint Conference on Neural Networks (IJCNN), Beijing, China, July 6-11, 2014. Results from this paper can be found in Chapter 4.

- E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.C. Liu, S. Furber, *Implementing Low-Latency, Energy-Efficient Spiking Deep Belief Networks on SpiNNaker*, Accepted in the International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, July 12-17, 2015. Results from this paper can be found in Chapter 5.

- E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.C. Liu, S. Furber, *Live Demonstration: Handwritten Digit Recognition Using Spiking Deep Belief Networks on SpiNNaker*, Accepted in the International Symposium on Circuits and Systems (ISCAS), Lisbon, Portugal, May 24-27, 2015. Results from this paper can be found in Chapter 5.

1.3.3 Workshops

Participation in various workshops enabled the author to collaborate with different groups across the globe and built a wide network of contacts. The workshops also had an important impact on the research presented in this thesis. Many early implementations and experiments described in this thesis were developed during these workshops, in particular:

- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2012*: Developed a closed-loop interface between a SpiNNaker Board and a Mobile Robot by utilising the on-board SpiNNaker links[3].
- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2013*: Interfaced SpiNNaker with a silicon cochlea and verified the correctness of the link through a series of experiments. Also, co-organised the *Running Neural Network Models on SpiNNaker* group[4].
- *Telluride Neuromorphic Cognition Engineering Workshop 2013*: Implemented spike-based Deep Belief Networks on SpiNNaker as part of the *Universal Neuromorphic Devices and Sensors for Real-Time Mobile Robotics* group[5]. Also, contributed to the *Sound Localisation* group[6]. The results from this workshop form parts of Chapters 4 and 5.
- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2014*: Optimised the performance of the spike-based Deep Belief Networks on SpiNNaker and compared results against a reference software simulator. The results from this workshop form parts of Chapter 5. Also, contributed to the *Deep Learning in computers, chips, and brains* discussion group[7].

1.4 Thesis Outline

The thesis consists of 6 chapters:

- **Chapter 2** presents how neurons and synapses can be mathematically modelled, and how computational frameworks can be developed by utilising them.

[3] https://capocaccia.ethz.ch/capo/wiki/2012/mmsnm12
[4] https://capocaccia.ethz.ch/capo/wiki/2013/spinnaker13
[5] http://neuromorphs.net/nm/wiki/DBNonSpiNNaker
[6] http://neuromorphs.net/nm/wiki/sound_localization
[7] https://capocaccia.ethz.ch/capo/wiki/2014/deeplearning14

- **Chapter 3** describes the different software tools and hardware platforms for simulating spiking neural networks and introduces the SpiNNaker platform.
- **Chapter 4** presents a methodology to characterise the power and latency of a SpiNNaker machine. Moreover, it presents a methodology to further reduce the overall energy consumption of larger SpiNNaker systems.
- **Chapter 5** investigates how hardware constraints impact the performance of spiking neural network implementations of deep belief networks. It also demonstrates two different topologies of such networks running on a SpiNNaker system in real-time and investigates the classification latencies, as a function of the input rates, and power requirements of larger networks.
- **Chapter 6** summarises the research presented in this thesis and suggests how it can be expanded in the future.

Chapter 2

Computing With Spiking Neural Networks

2.1 Introduction

Computational neuroscientists have been using simulations of biological tissue as an approach to understand how particular neural circuits work. Models of simulated neural networks, based on anatomical data, aim at reproducing phenomena observed in-vivo and in-vitro experiments. If the simulated network shows similar behaviour, it may then be used to describe how computations take place in that particular region. In addition, computational models provide greater observability and reproducibility. The overall benefit is that a sufficiently accurate model can be simulated repeatedly and in high-fidelity, without the noise of a biological recording and at whichever level of detail is required [Sejnowski, 2003]. Neurophysiologists often assume that all useful information about neural coding can be summarised in the form of Post-Stimulus Time Histogram (PSTH), which plots firing rate as a function of time. This assumption goes back to the late 1920s where Adrian [1928] showed that increasing the stimulus intensity of sensory fibers increased their firing rate. The work of Hodgkin and Huxley [1952] has given rise to the hypothesis that the precise time of action potential (spike) generation of a neuron, rather than simply its firing rate could be the information coding in the brain. Thorpe et al. [1996] demonstrated that humans respond selectively to complex visual stimuli so rapidly (in the order of 100-150 ms) that they argued that the time window is too small for rate coding [Thorpe et al., 2001]. Besides vision, other sensory pathways reveal fast processing; for example neurons in the bat auditory cortex can respond just 8 ms after the stimulus onset, which, based on the number of the subcortical

processing stages involved, leaves only a couple of milliseconds at each level [Jen et al., 1989]. Technological advancements in the computing industry allowed scientists to experiment with larger models. In the past, large-scale simulations of spiking neural networks were successfully executed on general-purpose supercomputers. The models used biological data for neural parameters and connectivity patterns and results revealed brain oscillations [Ananthanarayanan et al., 2009] and synchronisation between different cortical areas [Izhikevich and Edelman, 2008] similar to Functional Magnetic Resonance Imaging (fMRI) findings. Recently, Eliasmith et al. [2012] demonstrated the largest functional model simulation of a brain using biological characteristics and parameters estimated directly from neural data as constraints.

From an engineering point of view, SNNs, also referred to as the 3^{rd} generation of ANNs, are computationally more powerful than their predecessors [Maass, 1997] (rate-based neuron models) and with the development of the field of neuromorphic engineering [Mead, 1990; Liu et al., 2015] it has become possible to simulate neurons and synapses directly on hardware. This paves the way for the development of low-power, low-latency, defect-tolerant hardware circuits that are able to accelerate particular algorithms which require large amounts of resources on general purpose computing architectures [Merolla et al., 2014b; Monroe, 2014; Temam, 2012]. However, despite the computational power of SNNs they still lack the success of their predecessors and this is mainly because they lack sophisticated learning algorithms that have been developed over the years and have also benefited from the technological progress of conventional computers. To address this issue some research groups follow an intermediate step: a neural network is trained off-line using rate-based neuron models with state-of-the-art training algorithms [Hinton et al., 2006; LeCun et al., 1998b] and then the trained network is mapped to a SNN [O'Connor et al., 2013; Perez-Carrasco et al., 2013; Merolla et al., 2010], ready to be executed efficiently on a neuromorphic platform [Camuñas-Mesa et al., 2010; Arthur et al., 2012]. Training directly with spiking neurons is still an open research problem [Neftci et al., 2014].

The first part of this chapter aims to provide a brief biological background on neurons, synapses as well as different mathematical models used to describe them. The remainder of this chapter focuses on some of the current frameworks available for computations using SNNs. Both the neuron and synapse models as well as the spike-based computation frameworks presented here form just a small subset of those in the rich research environment of the neuromorphic community.

2.2 Biological Background

The neuron, which is the fundamental computational unit in the brain, is an electrically excitable cell that uses electrical and chemical signals to process and transmit information. Synapses are specialised connections that allow electrical or chemical signals to flow between neurons. Neural networks are formed by connecting neurons to each other.

As can be seen in Figure 2.1, a typical neuron consists of three distinct parts: the dendrites, the cell body (soma), and the axon. Dendrites are structures that emerge from the soma of a neuron and frequently reach out for hundreds of micrometers, branching numerous times, offering ascend to a complex *dendritic tree*. The axon is a long, slender cellular extension which starts from the soma, at a place known as the axon hillock, and extends for some distance (up to one meter in humans). The soma of a neuron may have several dendrites but only one axon, which can branch multiple times before it terminates. In most cases signals are transmitted from the axon to a dendrite through the synapse.

All neurons are electrically excitable, maintaining voltage gradients across their membranes by means of metabolically driven ion pumps, which combine with ion channels embedded in the membrane to generate intracellular-versus-extracellular concentration differences of ions such as sodium (Na^+), potassium (K^+), chloride (Cl^-), and calcium (Ca^{++}). Any alteration in the membrane voltage may modify the functionality of the voltage-dependent ion channels. An all-or-none electrical pulse, also known as action potential or spike, is generated every time the membrane voltage changes sufficiently. This spike will travel along the axon, activating the synaptic connections of other cells when it arrives [Dayan and Abbott, 2005; Gerstner and Kistler, 2002].

2.3 Models of Neurons and Synapses

2.3.1 Spiking Neuron Models

Spiking neuron models provide more biological fidelity than traditional sigmoidal units (rate-based). They take into account variations of the membrane potential caused by the opening and closing of ionic currents, and are modelled by a set of differential equations. If these equations model only a subportion of the neuronal membrane then these models are called multi-compartmental neuron models, otherwise if they model

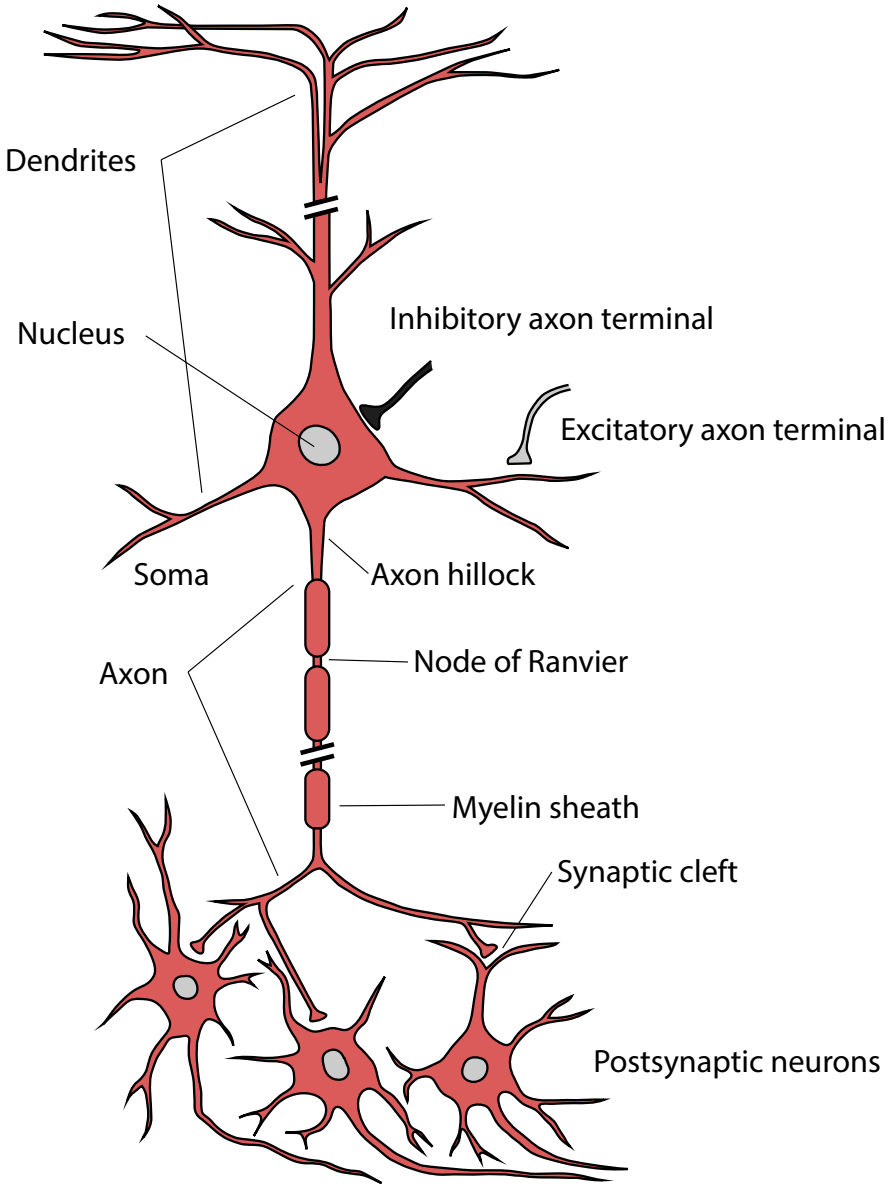


Figure 2.1: Simplified structure of a neuron. A neuron fires by transmitting electrical signals along its axon. When signals reach the end of the axon, they trigger the release of neurotransmitters that are stored in pouches called vesicles. Neurotransmitters bind to receptor molecules that are present on the surfaces of adjacent neurons. The point of virtual contact is known as the synapse [Trappenberg, 2005].

the whole neuron they are called single-compartment or point neuron models. This section first introduces the Hodgkin-Huxley model which provides a detailed model of the development of an action potential (spike) and then describes simplified models that are more suitable for large-scale simulations [Izhikevich, 2004].

2.3.1.1 Hodgkin-Huxley Model

Sir Alan Lloyd Hodgkin and Sir Andrew Fielding Huxley were the first to record an action potential from inside a giant squids nerve fibre in 1939 [Hodgkin and Huxley, 1939]. In 1952 they proposed the Hodgkin-Huxley (HH) neuron model [Hodgkin and Huxley, 1952] based on empirical data they obtained from experimenting with the squid giant axon. The HH model is based on the idea that the electrical properties of a segment of nerve membrane can be modelled by an equivalent circuit of the form shown in Figure 2.2. In the equivalent circuit, current flow across the membrane has two major components, one associated with charging the membrane capacitance and one associated with the movement of specific types of ions across the membrane. The ionic current is further subdivided into three distinct components, a sodium current I_{Na}, a potassium current I_K, and a small leakage current I_L that is primarily carried by chloride ions.

The behaviour of an electrical circuit of the type shown in Figure 2.2 can be described by a differential equation of the form:

$$I = C_m \frac{dV_m}{dt} - I_{ion} \tag{2.1}$$

where C_m is the membrane capacitance, V_m is the intracellular potential (membrane potential), I_{ion} is the ionic current flowing across the membrane, and I is an externally applied current.

The ionic current takes into account the contribution of the different ionic channels present in the membrane, and can therefore be subdivided according to the ion transported such as:

$$I_{ion} = I_{Na} + I_K + I_{leak} = g_{Na}(V - E_{Na}) + g_K(V - E_K) + g_{leak}(V - E_{leak}) \tag{2.2}$$

where I_{Na} and I_K represent the sodium and potassium ionic currents and I_{leak} represents a leakage current caused prevalently by Cl^- ions; the former two can be considered as time-variable conductances, while the latter is constant. E_{Na}, E_K and E_{leak} are called

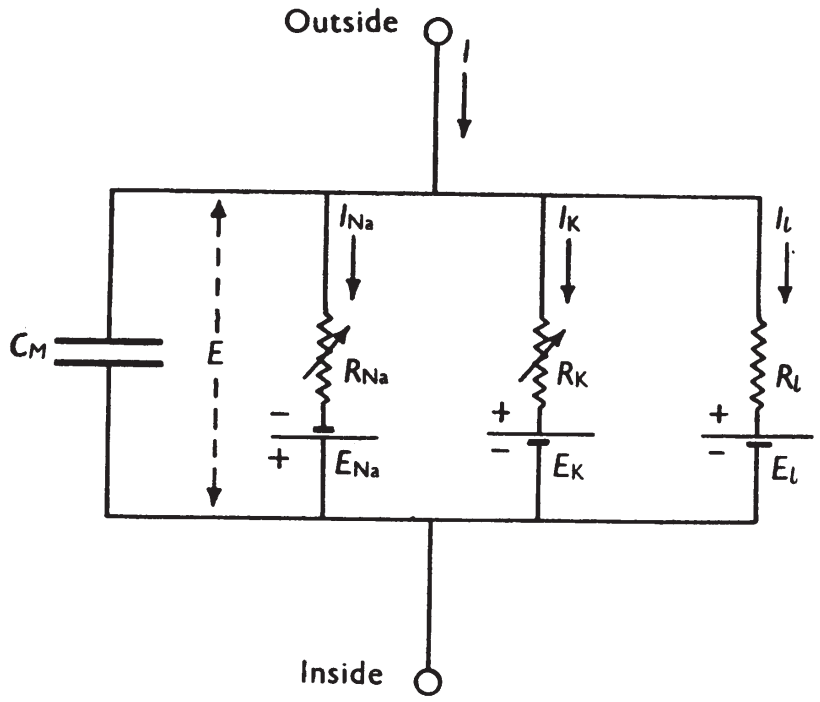


Figure 2.2: Electrical equivalent circuit proposed by Hodgkin and Huxley for a short segment of squid giant axon. The variable resistances represent voltage-dependent conductances [Hodgkin and Huxley, 1952].

reverse potentials, and represent the membrane potential at which the current flowing in a certain type of channels is equal to zero. The term $V - E_i$ is called driving force, and a current can be expressed as $g_i(V - E_i)$ [Dayan and Abbott, 2005].

Hodgkin and Huxley [1952] observed that the time- and voltage-dependent (in)activation of transmembrane channels varies membrane conductances and thereby gives rise to the action potential, which is ultimately described as:

$$C_m \frac{dV_m}{dt} = -\bar{g}_{Na} m^3 h(V - E_{Na}) - \bar{g}_K n^4 (V - E_K) - g_{leak}(V - E_{leak}) + I \qquad (2.3)$$

where $\bar{g}_{ion}$ is the maximum membrane conductance, m^3 may be viewed as the proportion of theoretical gating particles that are in an open state determining Na^+ conductance activation, h is a gating type variable that represents the level of inactivation, n^4 is the proportion of K^+-ion channels in the open state. $g_{leak}(V - E_{leak})$ denotes a leak current of Cl^-, for which conductance is constant. The differential equations for gating variables n, m, and h are expressed as:

$$\frac{dm}{dt} = \alpha_m(V)(1-m) - \beta_m(V)m, \tag{2.4}$$

$$\frac{dh}{dt} = \alpha_h(V)(1-h) - \beta_h(V)h, \tag{2.5}$$

$$\frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V)n, \tag{2.6}$$

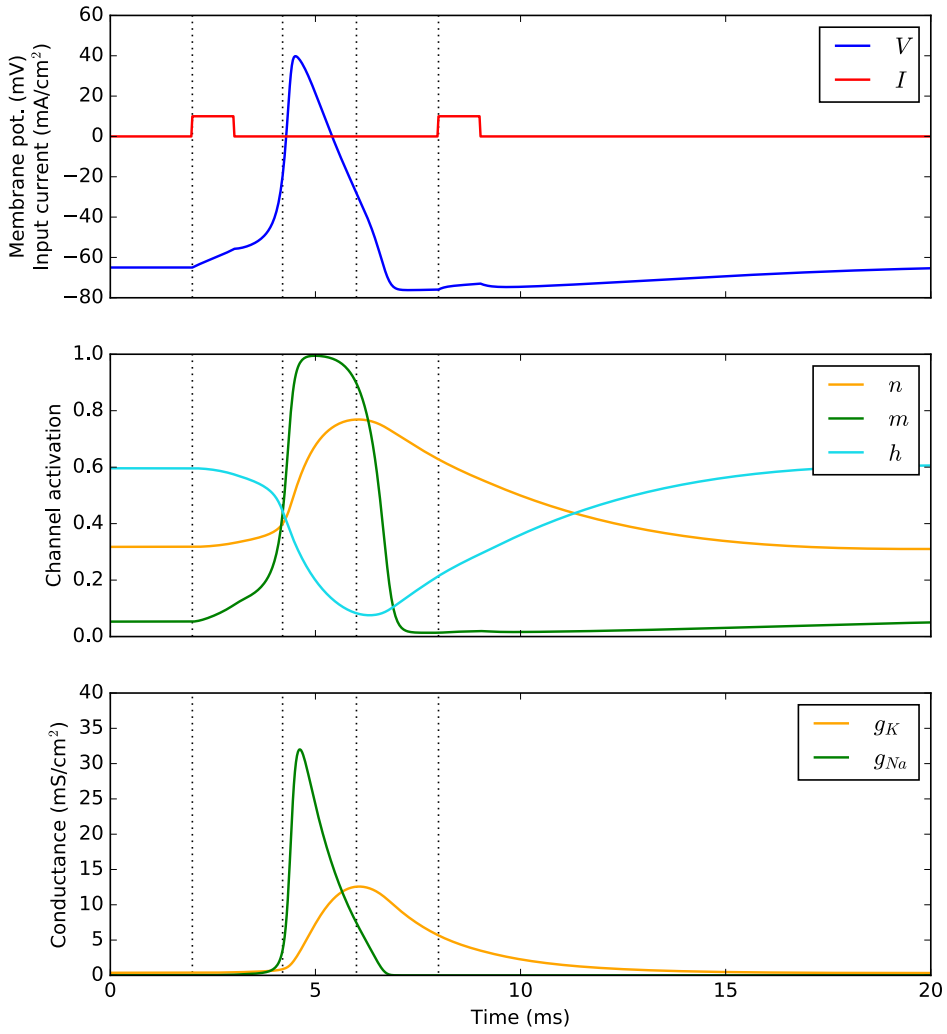


Figure 2.3: The time courses of membrane potential in the Hodgkin-Huxley model to a brief depolarising current. Conductances and gating variables during an action potential.

Figure 2.3 shows the generation of an action potential. Between $t = 0$ *ms* and $t = 2$ *ms* the membrane potential is in the resting state, the net sum of currents flowing through the membrane potential is 0. At $t = 2$ *ms* an input current is applied (either through synaptic inputs or through an electrode) which depolarises the membrane potential up to the Na equilibrium point E_{Na}. This results to an increase of opened Na gates, modelled by the m gating variable, that increases with V and further depolarises the membrane. As V increases Na gates are being inactivated by the h gating variable, which tends to 0, pushing the membrane towards the E_K equilibrium point. As a result there is a delayed activation (around $t = 6$ *ms*) of the K current that in conjunction with the inactivation of the Na channels, repolarises the membrane below the resting potential, also known as after-hyperpolarisation. As the gating variable h deactivates the Na conductance, the neuron is not capable of producing a subsequent action potential for some period of time, also known as the refractory period. This can be seen at around $t = 8$ *ms* where an input current of the same amplitude and duration as before does not trigger a subsequent action potential.

The HH neuron models the action potential in great detail taking into consideration the inactivation/activation of the Na and K channels. Simpler models can be constructed by treating action potentials as stereotypical events and using the time of their occurrence as the information transmitted. This gave rise to a number of simpler models described below.

2.3.1.2 Leaky Integrate-and-Fire Neuron

In the HH model the process of generating an action potential usually takes place when the membrane potential rises above the voltage-dependent Na channel equilibrium point. In addition, since action potentials are stereotyped events they are fully characterised by their firing time and not their shape [Gerstner and Kistler, 2002]. The Leaky Integrate-and-Fire (LIF) neuron model has the assumption that whenever the membrane potential reaches a threshold potential (V_{th}), an action potential is generated and the neuron is "reset" to a potential $V_{reset} < V_{threshold}$, taking into account the after-hyperpolarization of K^+ ions. The dynamics of the membrane potential of an LIF neuron can be described as:

$$c_m \frac{dV_m}{dt} = -g_{leak}(V - E_{leak}) + \frac{I}{A} \tag{2.7}$$

where A is the membrane area, $cm = Cm/A$ is the specific membrane capacitance, g_{leak}

is the leak conductance, E_{leak} is the equilibrium potential, and I is an external input current [Dayan and Abbott, 2005]. The equation may be multiplied through by the specific membrane resistance $r_m = 1/g_{leak}$ to give:

$$\tau_m \frac{dV_m}{dt} = E_{leak} - V + R_m I \tag{2.8}$$

where $\tau_m = c_m r_m$ is the membrane time constant and $R_m = r_m/A$ is the membrane resistance.

The advantages of the LIF neuron model is that it is much more computationally efficient compared to the HH model [Izhikevich, 2004], making it more suitable for large-scale simulations. In addition, it is a linear model and can therefore be treated analytically [Brunel, 2000]. However, because of its simplicity it can only replicate a small number of the known neuronal firing patterns [Izhikevich, 2004][1].

2.3.1.3 Izhikevich Neuron

Izhikevich [2003] created a model, which combines the dynamics of the HH model and the computational efficiency of the LIF model. This was achieved by reducing the 4 dimensional model of the HH model into two first order differential equations, as seen in Equations 2.9 and 2.10.

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - U + I(t) \tag{2.9}$$

$$\frac{dU}{dt} = a(bV - U) \tag{2.10}$$

$$\text{if } V \geq 30 \text{ mV then } \begin{cases} V = c \\ U = U + d \end{cases} \tag{2.11}$$

where the variable v represents the membrane potential of the neuron and u represents a membrane recovery variable, which models the activation of potassium K^+ ionic currents and inactivation of sodium Na^+ ionic currents. This model can exhibit all known

[1] Izhikevich [2004] categorised all known neuronal firing patterns, at that time, to 21 different classes, then compared a number of different neuronal models and showed that while the LIF is the most efficient to implement as it consists only of a single ordinary differential equation and one comparison operation (for the threshold) it can only replicate 3 out of the 21 different firing patterns. On the contrary, the HH model can reproduce all 21 firing patterns and offers high biological fidelity but requires a lot of floating point operations (1,200 compared to 5 for the LIF model) due to the number of differential equations that needs to be solved (see equations 2.2 - 2.6) and thus it is not suitable for large scale simulations.

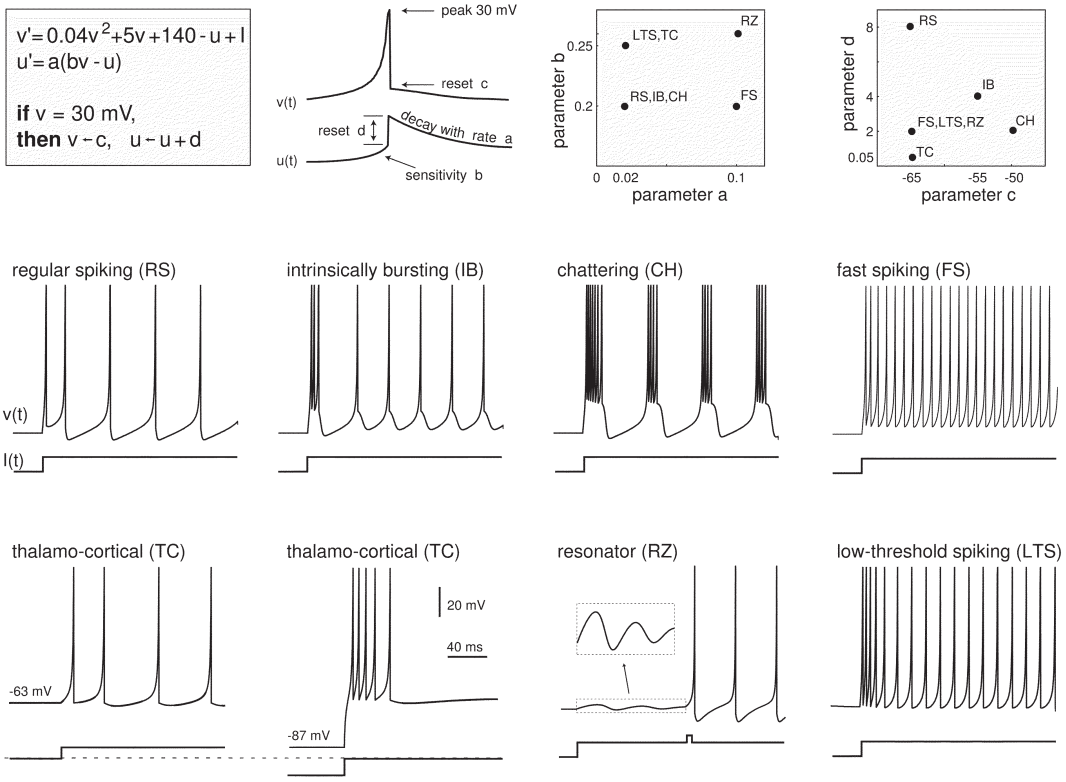


Figure 2.4: The Izhikevich neuron model. Known types of neurons correspond to different values of the parameters a, b, c, d in the model described by Equations 2.9, 2.10. RS, IB, and CH are cortical excitatory neurons. FS and LTS are cortical inhibitory interneurons. Each inset shows a voltage response of the model neuron to a step of dc-current $I = 10$ (bottom). Time resolution is 0.1 ms [Izhikevich, 2003].

neuronal firing patterns with the appropriate values for the a, b, c and d dimensionless variables [Izhikevich, 2004]. Furthermore, this model has a dynamic threshold that depends on the previous state of the membrane potential before the spike. In the next section the parameters of Equation 2.10 are explained [Izhikevich, 2003].

- The parameter a describes the time scale of the recovery variable u. Smaller values result in slower recovery. A typical value is $a = 0.02$.

- The parameter b describes the sensitivity of the recovery variable u to the sub-threshold fluctuations of the membrane potential v. A typical value is $b = 0.2$.

- The parameter c describes the after-spike reset value of the membrane potential v caused by the fast high-threshold K^+ conductance. A typical value for real neurons is $c = -65\ mV$.

- The parameter d describes the after-spike reset of the recovery variable u caused by slow high threshold Na^+ and K^+ conductance. A typical value is $d = 2$.

In Figure 2.4 the parameters a, b, c and d of the neuron model can be observed.

2.3.2 Models of Synaptic Transmission

Synapses are the specialised sites where one neuron communicates with another. There are two basic types of synaptic transmission: electrical or chemical. Electrical synapses can send rapid and stereotyped depolarising signals, while chemical synapses are capable of more variable signalling producing more complex behaviours. They can produce electrical changes to the postsynaptic cell that may last from milliseconds to minutes. Most synapses in the brain are chemical [Kandel et al., 1991].

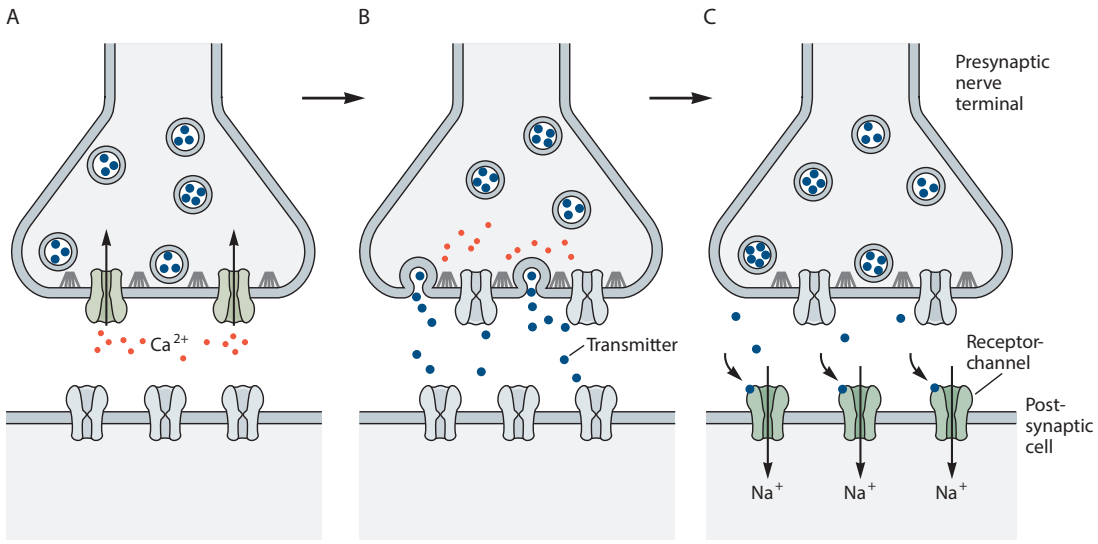


Figure 2.5: Chemical signalling in the synapse [Kandel et al., 1991].

Chemical synaptic transmission depends on the diffusion of a neurotransmitter across the synaptic cleft, a 20-40 nm gap between pre- and postsynaptic neurons. The neurotransmitters are kept within small membrane-bound spheres called vesicles. Upon the arrival of an action potential at the terminal of a presynaptic axon causes the voltage-gated Ca^{2+} channels to open, allowing an influx of Ca^{2+} to the presynaptic terminal (Figure 2.5(A)). The increase in intracellular Ca^{2+} concentration triggers a biochemical reaction that causes the vesicles to fuse with the presynaptic membrane and release neurotransmitter into the synaptic cleft (Figure 2.5(B)). The released neurotransmitter

molecules then diffuse across the synaptic cleft and bind specific receptors on the postsynaptic membrane. These receptors cause ion channels to open (or close), thereby changing the membrane conductance and membrane potential of the postsynaptic cell (Figure 2.5(C)). These processes can be modelled in a number of different ways described in the following subsections.

2.3.2.1 Instantaneous Rise and Single-Exponential Decay

This model assumes that the release of neurotransmitter, its diffusion across the cleft, the receptor binding, and channel opening all happen very quickly and the conductance g jumps to the peak conductance ($\bar{g}_{syn}$) at the time of the presynaptic spike (t_0) followed by an exponential decay with a time constant τ (Figure 2.6a):

$$g_{syn}(t) = \bar{g}_{syn} \exp(\frac{-(t-t_0)}{\tau}) \tag{2.12}$$

for $t < t_0$, $g_{syn}(t) = 0$. The differential equation describing the change of conductance in time is:

$$\tau \frac{dg_{syn}}{dt} = -g_{syn} + \bar{g}_{syn}\delta(t_0 - t), \tag{2.13}$$

where $\delta(t)$ is the Dirac delta function, and τ is the decay time constant which is determined by the unbinding of the neurotransmitter from the receptor channel.

2.3.2.2 Alpha Function

For some synapses the rising phase of synaptic conductances plays an important role on the network dynamics [Vreeswijk et al., 1994]. The alpha function describes a conductance that has a rising time (Figure 2.6b):

$$g_{syn}(t) = \bar{g}_{syn} \frac{t-t_0}{\tau} \exp(\frac{1-(t-t_0)}{\tau}) \tag{2.14}$$

for $t < t_0$, $g_{syn}(t) = 0$. However the rise and decay time courses are controlled by a single time constant (τ) and cannot be set independently.

2.3.2.3 Difference of Two Exponentials

The rise and decay phases can be expressed as the difference of two exponentials. This allows the time constants (τ_{rise}, τ_{decay}) to be set independently, so that for $t >= t_0$ (Figure 2.6c):

$$g_{syn}(t) = g_{max} f(\exp(\frac{-(t-t_0)}{\tau_{decay}}) - \exp(\frac{-(t-t_0)}{\tau_{rise}})) \tag{2.15}$$

where f is the normalisation factor to ensure that the amplitude equals $\bar{g}_{syn}$ [De Schutter, 2010].

2.4 Frameworks for Neural Computations

2.4.1 Biologically Plausible Synaptic Plasticity

Electrophysiological experiments have shown that the amplitude of the postsynaptic response to an incoming action potential of a synapse (also known as synaptic weight w_{ij}) is not fixed but can change over time. These changes can either be long-term or short-term. Forms of short-term plasticity include synaptic fatigue or depression and synaptic augmentation. Forms of long-term plasticity include long-term depression (LTD) when there is a decrease of the synaptic efficacy, and long-term potentiation (LTP) when there is an increase of the synaptic transmission efficacy. Synaptic plasticity is believed to be the basis of learning and memory in the brain.

Derived from biological observations that synaptic plasticity depends on the relative timing of pre- and post-synaptic spikes [Markram et al., 1997; Bi and Poo, 1998], Spike-Timing Dependent Plasticity (STDP) [Gerstner et al., 1996; Song et al., 2000] has become a popular model for learning in spiking neural networks. In its standard form, STDP weight-updates are expressed by the double-exponential form:

$$\Delta W = \begin{cases} A_+ exp(s/\tau_1) & \text{if } s < 0 \\ A_- exp(s/\tau_2) & \text{if } s \geq 0 \end{cases} \tag{2.16}$$

where $s = t_{\text{pre}} - t_{\text{post}}$ is the time difference between a pair of pre- and post-synaptic spikes, A_+ and A_- are scaling factors for potentiation and depression, and τ_+ and τ_- are the time constants of the plasticity curves. The weight update rule is illustrated in Figure 2.7. There are different strategies for computing the total amount of weight change after seeing multiple pre- and post-synaptic spikes [Morrison et al., 2008], e.g. by considering only nearest neighbour spike pairs, or summing the weight changes ΔW for all pairs.

The discovery of STDP has led to a number of models that have exploited the precise timing properties of spiking neurons for receptive field development [Clopath et al., 2010; Song and Abbott, 2001], temporal coding [Gerstner et al., 1996; Guyonneau et al.,

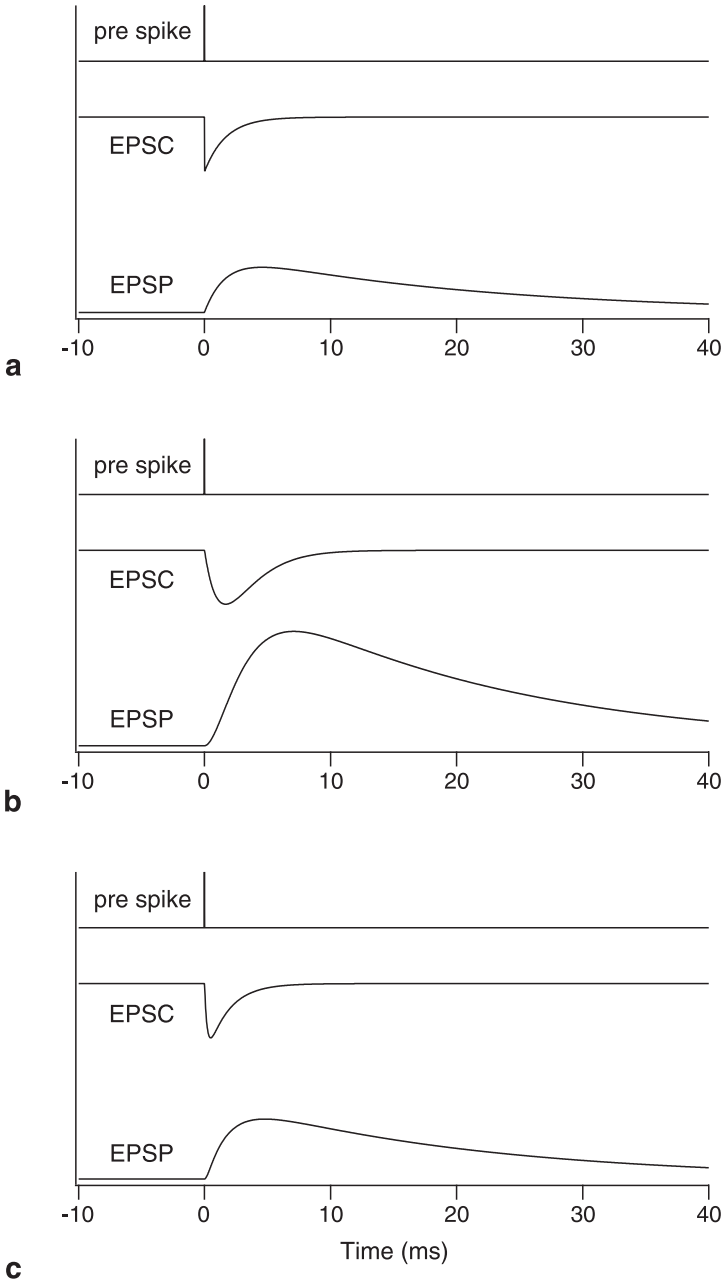


Figure 2.6: Synapse models: (a) Instantaneous Rise and Single-Exponential Decay, (b) Alpha Function, (c) Difference of Two Exponentials. For each model the variation of the synaptic conductance g_{syn} (EPSC - Excitatory Post Synaptic Current) and the effects of on the membrane potential (EPSP - Excitatory Post Synaptic Potential) to an incoming spike are shown [De Schutter, 2010]

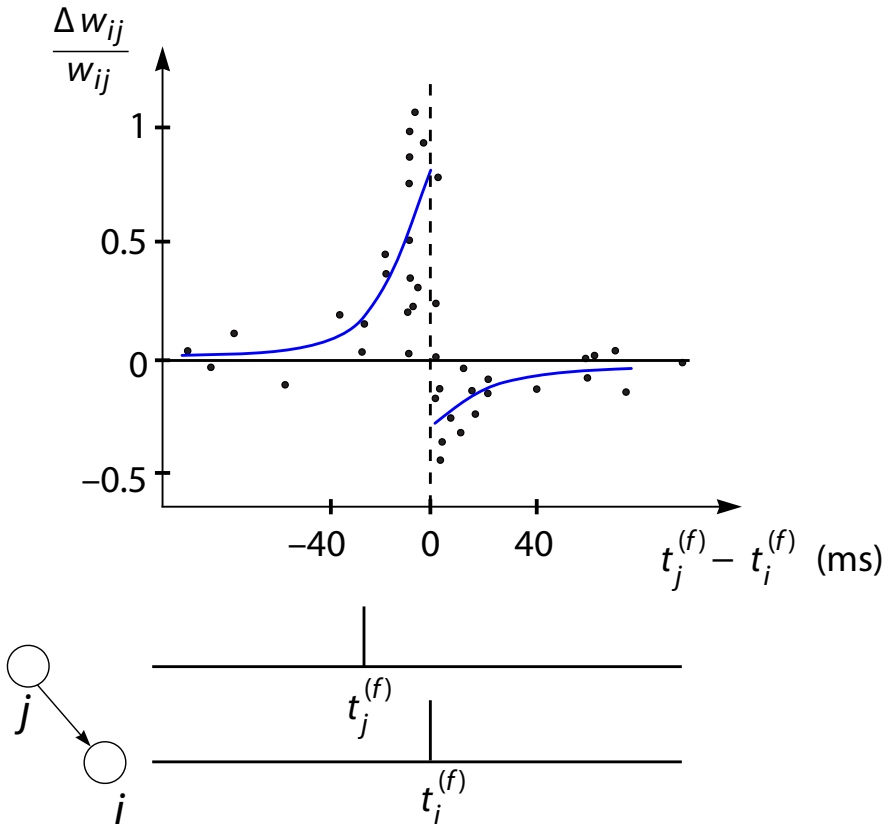


Figure 2.7: Timing requirements between pre- and postsynaptic spikes for Spike-Timing Dependent Plasticity (STDP). Synaptic changes Δw_{ij} occur only if presynaptic firing $t_j^{(f)}$ and postsynaptic activity at $t_i^{(f)}$ occur sufficiently close to each other. A positive change (LTP) occurs if the presynaptic spike precedes the postsynaptic one. The dots show experimentally measured weight changes as a function of $t_j^{(f)} - t_i^{(f)}$ overlayed on a fitted (solid line) two-phase learning window [Gerstner and Kistler, 2002].

2005], rate normalisation [Song et al., 2000; Kempter et al., 2001], or reward-modulated learning [Izhikevich, 2007; Legenstein et al., 2008; Friedrich et al., 2011; Potjans et al., 2011]. It has also been realized that there is not one standard model for STDP, but that there is a huge diversity of learning rules in nature, depending on species, receptor, and neuron types [Abbott and Nelson, 2000; Kullmann et al., 2012], the presence or absence of neuromodulators [Pawlak et al., 2010; Cassenaer and Laurent, 2012], but also on other factors like post-synaptic membrane potential, position on the dendritic arbor, or synaptic weight [Sjöström et al., 2001].

STDP can be seen as a spike-based formulation of a Hebbian learning rule, which says that synaptic connections strengthen when two connected neurons have correlated firing activity. This formulation suggests a potential causal relation between the firing of the two neurons. Causality requires that the presynaptic neuron fires slightly before the postsynaptic one. Indeed, in standard STDP experiments on synapses onto pyramidal neurons, potentiation of the synapse occurs for pre-before-post timing, in agreement with Hebbs postulate. Other popular learning rules include the **B**ienenstock, **C**ooper, and **M**unro (BCM) rule [Bienenstock et al., 1982], which was developed to explain the development of receptive field properties of neurons in primary sensory cortex.

2.4.2 Liquid State Machines

The neocortex comprises 80% of the human brain and is arranged in repeating stereotypical neural microcircuits, also known as cortical columns that have approximately 0.3 mm diameter and 2 mm depth [Kandel et al., 1991]. Each of these microcircuits can participate in a number of different tasks concurrently. In addition, it is thought that neural microcircuits form very high-dimensional dynamical systems where each neuron and synapse adds additional degrees of freedom to the dynamics of the system [Maass et al., 2002b].

Wolfgang Maass et al. [2002b] proposed the Liquid State Machine (LSM) as a computational framework to explain how computations of continuous streams of multi modal inputs from a rapidly changing environment could be taken place in generic cortical microcircuits. In their proposed approach instead of trying to control the dynamics of a recurrent neural network, as for example in attractor neural networks to produce stable outputs, a readout is used to extract a stable output even if the liquid (the internal state of the microcircuit) never revisit the same state.

The idea of feeding an input signal into a fixed random recurrent neural network, also known as *reservoir*, and then training a simple *readout* mechanism to read the state

of the reservoir and map it to the desired output is part of a computational paradigm known as Reservoir Computing (RC) [Verstraeten et al., 2007]. The main benefit of RC is that the training is performed only at the readout stage which can be any type of classifier or regressor, from perceptrons [Maass et al., 2002b] to SVMs [Boser et al., 1992], while the reservoir is randomly generated and remains fixed. This way the reservoir functions as a kernel as in the case of kernel-based methods, by projecting the inputs into a high-dimensional space (a method known as the *kernel trick*) which enhances the separability. LSMs and Echo State Networks (ESNs) [Jaeger, 2001] are two major types of reservoir computing [Verstraeten et al., 2007].

In the case of LSMs the connectivity and parameters of the liquid (reservoir) are inspired by biological findings, e.g. the rat's somatosensory cortex [Gordon Shepherd and Grillner, 2010; Markram et al., 1998]. Instead of using analogue sigmoidal neurons, as in the ESNs, more sophisticated neuron models are used, e.g. the LIF model that allow more complicated spatio-temporal information processing [Maass, 1997]. Similarly to ESNs, a readout can learn to extract information from the high-dimensional transient states of the reservoir without the need of the neural microcircuit to ever reach a stable state.

The term liquid comes from the idea that the recurrent connectivity of the spiking neurons in the reservoir cause the input signals to remain present for some time, a situation resembling ripples created when throwing a stone in a pond.

In the original proposal [Maass et al., 2002b,c] the liquid was constructed as a three dimensional grid cortical column using LIF neurons, where 20% of them were inhibitory and 80% excitatory. The probability of two neurons a and b being connected is given by equation 2.17:

$$P_{conn(a,b)} = Ce\frac{-D(a,b)}{\lambda} \tag{2.17}$$

where variable λ controls both the average number of connections as well as the average synaptic distance of the neurons, $D(a,b)$ is the Euclidean distance between neurons a and b, while the value of C depends if neurons a and b are excitatory or inhibitory.

Maass et al. [2002b] identified two conditions that have to be met in order to achieve powerful real-time computations. The first one regards the ability of the liquid to identify two different signals, known as separation property (SP), while the second one regards the ability of the readout to have adequate resolution and recording capabilities to distinguish and transform the internal state of the reservoir to given target outputs, known as approximation property (AP).

The LSM, as seen in Figure 2.8, can be expressed in a mathematical form where the liquid (reservoir) is a filter L^M that projects an input signal $u(t)$ onto the reservoir state $x(t)$, where $x(t)$ depends not only on the current $u(t)$ but also in previous inputs $u(s)$, where $(t > s)$:

$$x(t) = (L^M u)(t) \tag{2.18}$$

where the filter $L^M u(t)$ is the output of filter L at time t when L is applied to an input signal $u(\cdot)$. When the filter L^M is implemented using spiking neurons is called liquid neurons.

The readout is a memoryless readout function f^M that extracts the current state of the liquid $x(t)$ into a target output, at every time t, equation 2.19.

$$y(t) = f^M(x^M(t)) \tag{2.19}$$

Memoryless means that the readout function f^M does not need to retain memories from past states of the liquid since they are already contained in the current state. However the readout usually has synapses that are trained to a specific task and in that sense it contributes to the total memory capabilities of the system [Maass et al., 2002b].

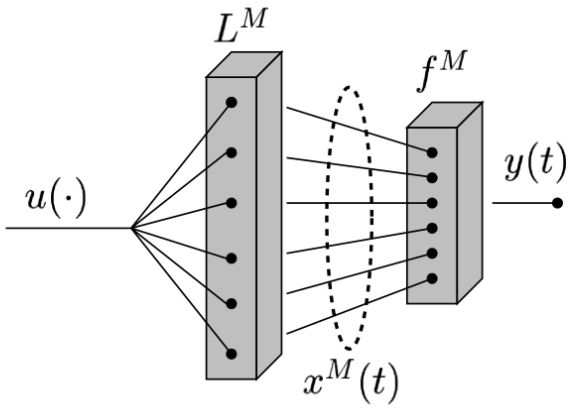


Figure 2.8: A Liquid State Machine architecture. Where $u(\cdot)$ is a continuous input signal injected to the reservoir (liquid filter f^M), creating a liquid state $x(t)$. Finally, a readout is used to map the f^M to a target output [Maass et al., 2002b].

The value of λ, affects the separation property of the liquid. It has been demonstrated that even though large values of λ increase the separation property of the liquid, the average correctness of the output is reduced. This indicates that by increasing the

separation property too much it also increases the sensitivity to noise [Maass et al., 2002b]. Finally, the separation property can be further improved by adding additional cortical columns without modifying the previous ones, as seen in Figure 2.9.

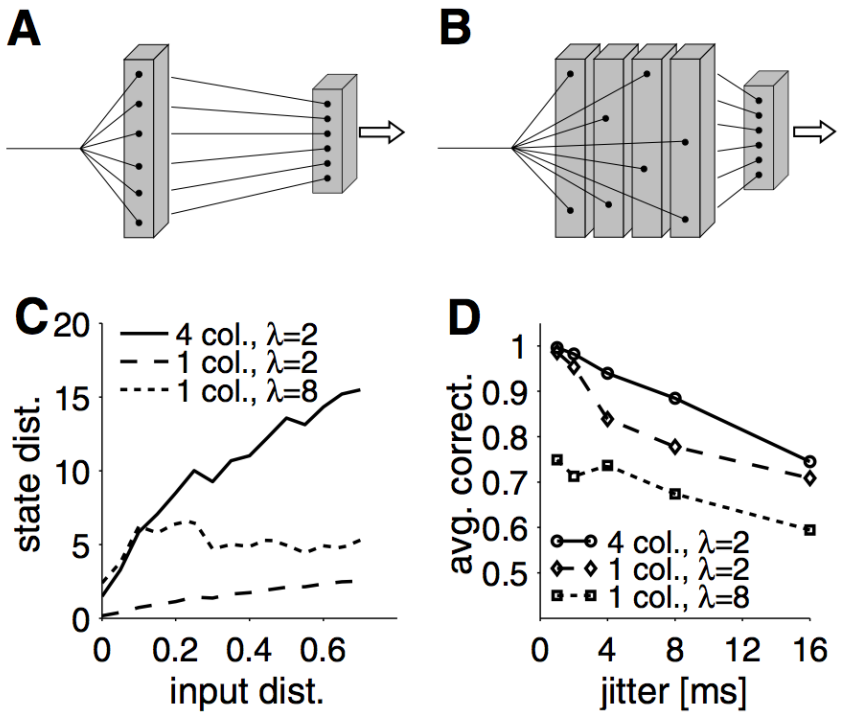


Figure 2.9: Separation property and performance of liquid circuits. A comparison between single (A) and multiple liquids (B) against the separation property (C) and noise sensitivity (D) [Maass et al., 2002b].

LSMs have been applied to tasks with strong temporal aspects. On the Hopfield & Brody [Hopfield and Brody, 2000, 2001] speech recognition task a LSM was able to achieve a lower average square error with much fewer neurons [Maass et al., 2002b], while the performance increased even more in the presence of noise. Verstraeten et al. [2005] also demonstrated that a LSM was more robust to input noise compared to a state-of-the-art recognition system on a speech recognition task. Finally, LSMs have been successfully utilised in movement prediction [Maass et al., 2002a] and robotic arm control [Joshi and Maass, 2005; Probst et al., 2012] tasks.

2.4.3 Polychronization

Polychronization is a concept described by Izhikevich [2006] that takes advantage of the axonal conductance delays [Swadlow, 1985] along with synaptic plasticity to organise and fine tune postsynaptic neurons in order to respond to certain time-locked input patterns. Polychronization is the ability of an SNN to exhibit reproducible time-locked, but not synchronous, firing patterns with millisecond precision. Based on the connectivity between neurons, a polychronous group (PG) is a possible stereotypical time-locked firing pattern.

An example neural network demonstrating the concept of polychronization can be seen in Figure 2.10A. In this network the presynaptic neurons connect to the postsynaptic neurons with different axonal conductance delays. More specifically, neuron *a* receives inputs from neurons *b*, *c*, and *d* with 1 ms, 5 ms and 9 ms axonal delays, while neuron *e* receives inputs from the same presynaptic neurons but with 8 ms, 5 ms and 1 ms delays. If all presynaptic neurons fire synchronously, as depicted in Figure 2.10B, the postsynaptic neurons *a* and *e* will not fire because the presynaptic spikes will arrive at different times. If the presynaptic neurons fire at a temporal pattern determined by the delays and shown in Figure 2.10C then spikes will arrive at the same time to neuron *a* causing it to fire a spike but will have no effect on neuron *e*. A different temporal fire pattern as shown by Figure 2.10D excites neuron *e* but not neuron *a*.

Izhikevich [2006] demonstrated that in a randomly connected SNN neurons self-organize spontaneously into groups and generate firing patterns of stereotypical polychronous activity, due to the delays and STDP synaptic plasticity. One advantage of this method is that because each neuron can be part of many polychronous groups, the total number of polychronous groups far exceed the total number of neurons in the network [Izhikevich, 2006], which means that the representational power of the neural network is increased.

Izhikevich and Hoppensteadt [2009] described how the concept of polychronization could be used as a computational framework termed Polychronous Wavefront Computation. The Polychronous Wavefront Computation is an event-driven framework where information is encoded in time. Computations are based on transponders, which are neuron-like devices, and on pulses of temporal and spatial patterns of activity, that cause transponders to create pulses in response to coincident inputs. Pulses are propagated as circular waves from the sources to other transponders. Computations result from interactions between transponders, and they are encoded by the exact physical locations of transponders and by precise timings of pulses [Hart, 2014]. Finally, a number of

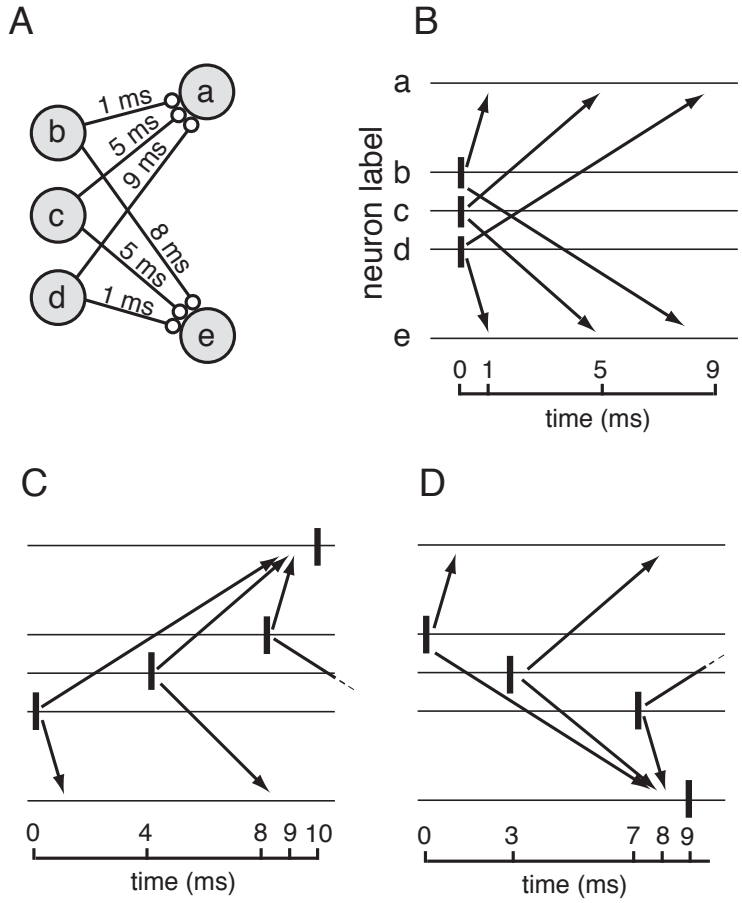


Figure 2.10: An example neural network demonstrating the concept of polychronization. (A) Presynaptic neurons connect to postsynaptic neurons with different axonal conduction delays. (B-D) Spikes are denoted as vertical bars, while the arrows represent the time arrival of the spike to the postsynaptic neuron. (B) Synchronous firing of the presynaptic neurons does not cause the postsynaptic neurons to fire because spikes arrive at different times. (C) When the presynaptic neuron *b* fires at 0 ms, neuron *c* fires at 4 ms and *d* fires at 8 ms, all presynaptic spikes arrive at the postsynaptic neuron *a* simultaneously causing it to fire a spike, while they have no effect on neuron *e*. (D) The reverse order of firing is optimal to excite neuron *e* [Izhikevich, 2006].

groups have investigated the use of a polychronous layer as the liquid in LSMs [Galluppi and Furber, 2011; Paugam-Moisy et al., 2008].

2.4.4 Neural Engineering Framework

The Neural Engineering Framework (NEF) [Eliasmith and Anderson, 2004] provides a set of methods for building biologically plausible models based on a functional specification of a neural system. The main idea of NEF is that a vector within a particular space can be represented by a population of spiking neurons and that connections between populations can compute functions on those vectors. The NEF provides a set of tools and methods on calculating what the connections need to be in order to compute a desired function on the vector space represented by a group of neurons. NEF utilises three basic principles: representation, transformations and neural dynamics. More specifically:

- **Representation:** A group of neurons represents a vector of a specific length (e.g. a 2-dimensional vector). This generally uses a distributed encoding, and is highly non-linear due to the inherent neuron non-linearities. However, a linear decoding on the spiking output of the group of neurons can be used to accurately recover the original input.

- **Transformation:** A connection from one neural population to another computes a function on the represented value, so if the first neural population represents x, then the second neural population represents $f(x)$. An arbitrary function may be chosen and then solve for the connection weights that will approximate that function. The approximation will be more accurate the more neurons are used, and less accurate the more non-linear the function.

- **Dynamics:** Recurrent connections allow us to define complex dynamical models. By adapting a standard control theory framework, we can implement integrators, Kalman filters, and other useful reactive systems.

Neural representations are defined by the combination of non-linear encoding and weighted linear decoding. For the encoding part, an analogue value is translated to a spike train for each neuron i in a neuronal population using the following equation:

$$\sum_n \delta(t - t_{in}) = G_i[\alpha_i \langle \tilde{\phi}_i \mathbf{X}(t) \rangle_m + J_i^{bias}] \tag{2.20}$$

where $\delta(t-t_{in})$ is the spiking activity of neuron i with each spike indexed by n, G_i is a non-linear function expressed as the neuron model, α_i is a gain factor, $\mathbf{X}$ is the vector signal to be encoded, $\tilde{\phi}$ is the encoder and J^{bias} is a bias current. The encoding process can be thought of as the preferred direction (tuning curve) [Georgopoulos et al., 1986; Butts and Goldman, 2006] of a neuron to a specific input stimulus.

An estimate of the original signal is recovered through linear population decoding, using the vectors ϕ and linear temporal decoding using the filter $h(t)$, which can be thought of as a simple model of post-synaptic current (PSC) (Section 2.3.2). These can be combined to give a population-temporal decoder, $\phi_i(t-t_{in})$:

$$\hat{\mathbf{X}}(t) = \sum_i^N \alpha_i(\mathbf{X}(t))\phi_i^{\mathbf{X}}, \tag{2.21}$$

where

$$\begin{aligned} \alpha_i(\mathbf{X}(t)) &= \sum_i h_i(t) * \delta(t-t_{in}) \\ &= \sum_i h_i(t-t_{in}) \end{aligned} \tag{2.22}$$

Transformations of neural representations are functions of variables that are represented by neural populations. Transformations are determined using an alternately weighted linear decoding (i.e., the transformational decoding as opposed to the representational decoding).

Quantitatively, we assume the same encoding as described in principle 1 (representation) and define the decoding:

$$\hat{f}(\mathbf{X}(t)) = \sum_i \alpha_i(\mathbf{X}(t))\phi_i^f \tag{2.23}$$

This decoding is the similar to that in Equation 2.21, except the decoders are determined such that a function, f, of the original input signal is estimated. The connection weights can be calculated knowing the decoders of the source population and the encoders of the target population (substituting Equations 2.20 and 2.21):

$$w_{ij} = \alpha_j \phi_i^{\mathbf{X}} \tilde{\phi}_j \tag{2.24}$$

where i is the index of the neuron of the presynpatic population, and j those in the postsynaptic population. The accuracy of the computation depend on the total number

of neurons of the two populations [Eliasmith, 2005].

Neural dynamics are characterised by considering neural representations as control theoretic state variables. Thus, the dynamics of neurobiological systems can be analysed using control theory, thus the general expression for the encoding described by Equation 2.20 can be rewritten as:

$$\sum_n \delta(t-t_{in}) = G_i[\alpha_i\langle\tilde{\phi}_i(h_i(t)*[\mathbf{A}'\mathbf{X}(t)+\mathbf{B}'\mathbf{u}(t)])\rangle_m + J_i^{bias}] \tag{2.25}$$

where $\mathbf{A}'$ is the neural dynamics matrix, and $\mathbf{B}'$ is the input matrix. These matrices define the dynamics of the system, and can be related to the standard dynamics and input matrices in linear control theory using:

$$\begin{aligned}\mathbf{A}' &= \tau\mathbf{A}+\mathbf{I}\\ \mathbf{B}' &= \tau\mathbf{B}\end{aligned} \tag{2.26}$$

where the signal $u(t)$ is the input, $x(t)$ is the neural population's represented state vector, and τ is the time constant of the PSC (exponential function described in Section 2.3.2). For recurrent connections Equation 2.24 becomes:

$$w_{ij} = \langle\phi_i^{\mathbf{X}}\mathbf{A}'\tilde{\phi}_j\rangle \tag{2.27}$$

The NEF has been successfully utilised to model a number of neural systems ranging from motor control, sensory processing, to a model of basal ganglia that showed reaction times and error rates comparable to the ones of human subjects [Eliasmith and Anderson, 2004]. Its biggest success so far, however, is the Semantic Pointer Architecture Unifed Network (Spaun) [Eliasmith et al., 2012], which is the largest model simulation of a functional brain to date. Spaun consists of 2.5 million spiking neurons divided into several cranial subsystems, including the prefrontal cortex, basal ganglia, and thalamus, wired together to mimic the wiring of a human brain. Spaun is capable of performing a series of cognitive tasks using the same architecture and parameters. The basal ganglia (action selection system) dynamically recruits the appropriate neural subsystems required to solve a particular task. The anatomical and functional architecture of Spaun can be seen in Figure 2.11.

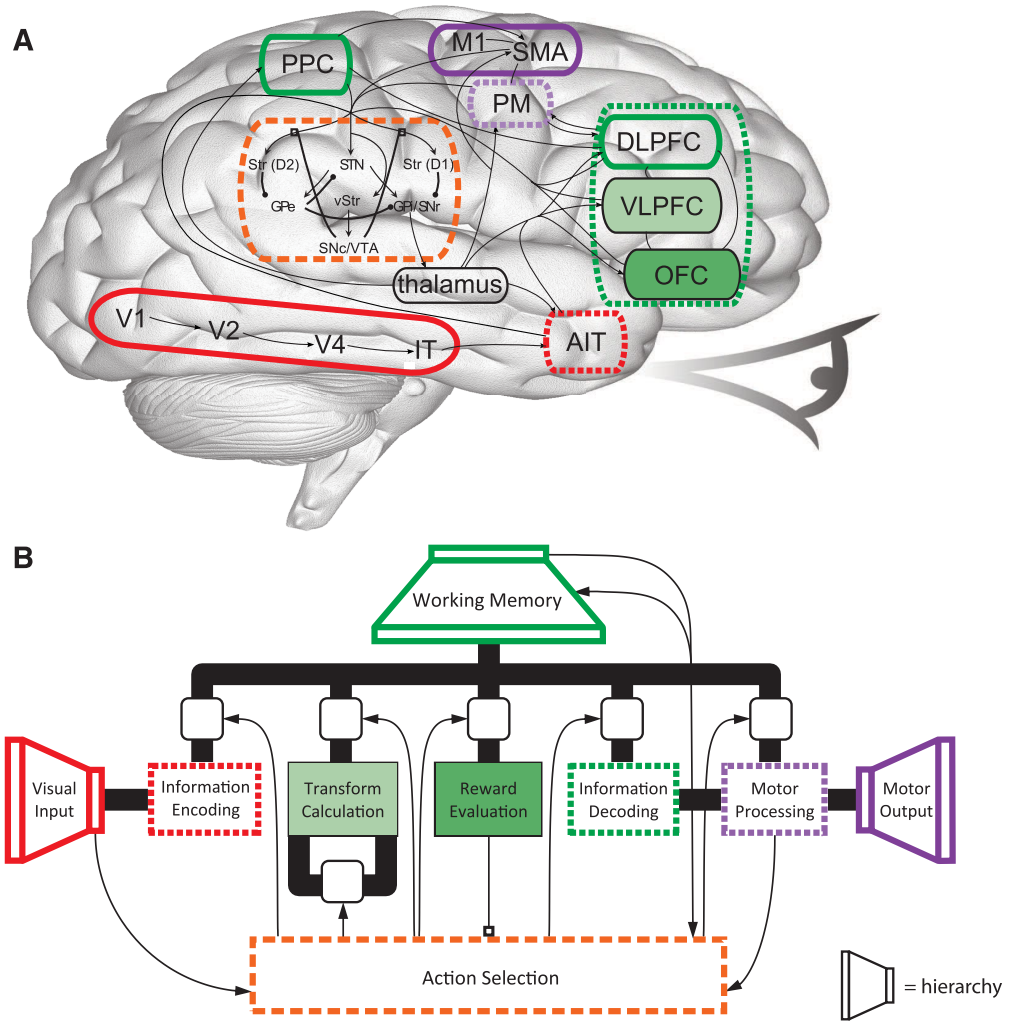


Figure 2.11: Anatomical and functional architecture of Spaun. (A) The anatomical architecture of Spaunn presents the brain structures included in the model, where PPC stands for posterior parietal cortex; M1 for primary motor cortex; SMA for supplementary motor area; PM for premotor cortex; VLPFC for ventrolateral prefrontal cortex; OFC for orbitofrontal cortex; AIT for anterior inferior temporal cortex; Str for striatum; vStr for ventral striatum; STN for subthalamic nucleus; GPe for globus pallidus externus; GPi for globus pallidus internus; SNr for sub- stantia nigra pars reticulata; SNc for substantia nigra pars compacta;VTA for ventral tegmental area;V2 for secondary visual cortex; V4 for extrastriate visual cortex. Box styles and colors indicate the relationship with the functional architecture. (B) The functional architecture of Spaun. The thick black lines represent communication between the elements of the cortex, while the thin black lines indicate communication between the action-selection mechanism (basal ganglia) and the cortex. The open-square end of the line connecting reward evaluation and action selection denotes that this connection modulates connection weights [Eliasmith et al., 2012].

2.4.5 Deep Learning Architectures

Deep learning architectures [LeCun et al., 2015; Schmidhuber, 2015] , which consist of Convolutional Networks [LeCun et al., 1998b], Deep Autoencoders [Hinton and Salakhutdinov, 2006], and Deep Belief Networks (DBNs) [Hinton and Salakhutdinov, 2006] is a branch of machine learning that currently represent the state-of-the-art solution for virtually all relevant machine learning tasks including computer vision [Larochelle et al., 2007; Lee et al., 2009; Cireşan et al., 2010; Le et al., 2012; Schmidhuber, 2015], and speech recognition benchmarks [Dahl et al., 2012; Hinton et al., 2012; Mohamed et al., 2012] and have thus been named one of the breakthrough technologies of the decade [Hof, 2013], leading to what has been called the "second reNNaissance of neural networks" [Cireşan et al., 2010]. The performance of these networks can be increased by increasing the size of the networks, i.e. using networks with more layers, as described by theoretical results showing that adding more layers can only improve performance bounds [Hinton et al., 2006].

Networks with large number of neurons and layers have very high computational demands, and training state-of-the-art deep networks can easily take multiple days, even on very large computer clusters [Dean et al., 2012], therefore calling for hardware accelerations, either through GPUs, or custom chips [Farabet et al., 2011]. Even the execution of a trained network on standard PCs is expensive due to the large number of neurons involved, and results in high energy demands, communication overheads, and high response latencies. In particular, the long latency response is a problem for real-time applications in mobile and robotic systems, which have limited computing resources and power but require quick system responses.

One way to overcome the aforementioned issues is to transform these type of networks into spiking neural networks which can be optimally implemented on event-based neuromorphic hardware platforms [Indiveri et al., 2011] that provide low-latency and energy efficient solutions. This has been recently demonstrated by O'Connor et al. [2013] for DBNs and by Perez-Carrasco et al. [2013] for Convolutional Networks and will be described in the following sections.

2.4.5.1 Convolutional Networks

Convolutional Networks are a variation of multilayer feed-forward neural networks inspired by the findings of Hubel and Wiesel [1959], on the structure of the early stages of visual cortex in mammals, and have been used extensively for image processing

and machine vision tasks. A predecessor to convolutional networks was introduced by Fukushima [1980] named *neocognitron*, while the model was further improved by LeCun et al. [1998b].

Typically, a Convolutional network is composed of alternating layers of convolution and spatial subsampling, with nonlinearities between subsequent iterations. Convolutional Networks introduce three basic ideas: local receptive fields, shared weights, and pooling.

Neurons in a convolutional layer are connected only to a subregion of the layer before it (local receptive field), instead of all neurons as in a fully-connected manner (e.g. MLPs). This local receptive field slides (convolves) for all neurons in this convolutional layer and all neurons in this convolutional layer share the same weights, meaning that they detect exactly the same features. This is why the hidden layer is also known as feature map. A complete convolutional layer consists of several different feature maps also known as filters or kernels, which are smaller than the dimension of the previous layer, as seen in Figure 2.12. Some of the advantages of the shared weights per kernel is that they greatly reduce the number of learning parameters, which also reduces the memory requirements, and also results in speed-ups during the training process when compared to fully-connected neural networks [LeCun et al., 1998b].

A pooling layer is added periodically in-between successive Convolutional layers. A popular pooling method is the MAX operation. MAX-pooling partitions the input region of the feature map into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum activation. This way MAX-pooling acts non-linear down-sampling that reduces the spatial size of the representation, the computations for the upper layers and finally provides a form of translation invariance. Besides MAX-pooling, the pooling layer may implement other functions, such as average pooling or even L2-norm pooling.

The top layer of a Convolutional network consists of a fully-connected neural network that receives input from the layer below (pooling layer). Convolutional networks are trained using the stochastic gradient descent and backpropagation [Werbos, 1974] with a few modifications since backpropagation was designed for fully-connected neural networks.

Perez-Carrasco et al. [2013] proposed a method for mapping the parameters of a Convolutional network properly trained with rate-based (sigmoidal) neurons into the equivalent event-driven (spiking) neurons by scaling the learnt weights. Their method was tested on a silhouette orientation recognition and poker card symbol recognition task,

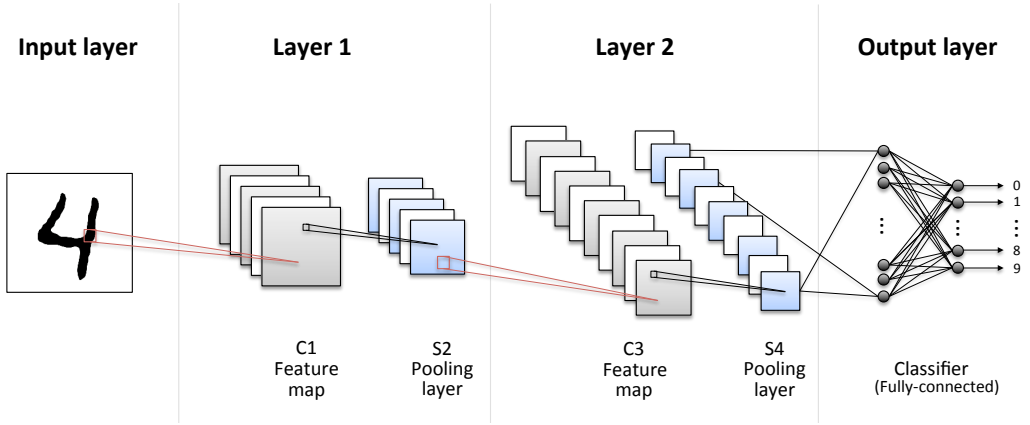


Figure 2.12: Topology of a typical Convolutional Network. The first layers compute an increasing number of feature maps with decreasing resolution by convolution with $n \times n$ kernels and subsampling. At the higher layers, the resolution drops to 1×1 and the weights are fully connected.

using inputs from a biologically inspired asynchronous vision sensor [Leñero-Bardallo et al., 2011] (described in Chapter 3). Results showed the high speed response capability of event-driven sensing and processing systems, as recognition is achieved while the vision sensor is delivering its output, a feature also termed as "*pseudo-simultaneity*" [Perez-Carrasco et al., 2013; Camuñas-Mesa et al., 2010].

2.4.5.2 Deep Belief Networks

Whereas Convolutional Networks currently outperform other architectures on many vision tasks, the alternative architecture of DBNs [Hinton and Salakhutdinov, 2006] remains very popular due to its ability to learn from large unlabelled datasets [Le et al., 2012], and because of its dual role as classifier and generative model of the data. In addition, DBNs have been shown to improve theoretical performance bounds by adding additional layers of neurons [Hinton and Salakhutdinov, 2006].

DBNs are multi-layered neural networks, in which each layer pair is formed by a Restricted Boltzmann Machine (RBM), a recurrent network with full connectivity between two layers of visible and hidden units, but without connections between neurons of the same layer, as seen in Figure 2.13. Each neuron unit is a stochastic binary neuron, whose "on" probability is given by the weighted sum of its inputs and passed through a sigmoid nonlinearity. Training is performed layer-by-layer using the unsupervised Contrastive Divergence (CD) rule [Hinton and Salakhutdinov, 2006]. After training one layer, the outputs of the hidden units of the previous layer are used as the input to the

subsequent layer. At the topmost level, a label is jointly trained with the input to give a supervised signal that guides the output of the network.

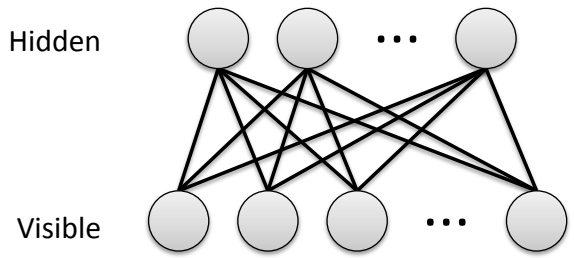


Figure 2.13: Architecture of a single Restricted Boltzmann Machine with full connectivity between visible units (bottom) and hidden units (top), but no connections within the same layer.

Training of DBNs targeting a spiking network implementation is described in detail in O'Connor et al. [2013]. The following section provides a brief summary of the most important differences to conventional CD training. The key idea is to use an accurate rate-based approximation of the firing rates of LIF neurons, and translate this into activation probabilities, which can be used in the CD updates. For this reason the so-called Siegert approximation [Jug et al., 2012] is being used, which approximates the output firing rate of a LIF neuron receiving both inhibitory and excitatory inputs. Let $\vec{\rho}_i$ and $\vec{\rho}_e$ be the vectors of inhibitory and excitatory input rates, and $(\vec{w}_i, \vec{w}_e)$ be the corresponding weights. In order to compute the expected output rate of the LIF neuron, a number of auxiliary variables first needs to be computed. For completeness, the full equations are provided here, but refer to previous work for the derivation and interpretation of each variable [Siegert, 1951; Jug et al., 2012]:

$$\begin{aligned} \mu_Q &= \tau \sum (\vec{w}_e \vec{\rho}_e + \vec{w}_i \vec{\rho}_i) & \sigma_Q^2 &= \frac{\tau}{2} \sum (\vec{w_e^2} \vec{\rho}_e + \vec{w_i^2} \vec{\rho}_i) \\ \Upsilon &= V_{\text{reset}} + \mu_Q & \Gamma &= \sigma_Q \\ k &= \sqrt{\tau_{\text{syn}}/\tau} & \gamma &= |\zeta(1/2)| \end{aligned}$$

Here, τ_{syn} denotes the synaptic time constant (which is considered to be zero), and ζ is the Riemann zeta function. Then the average firing rate ρ_{out} of the neuron with reset potential V_{reset}, threshold voltage V_{thresh}, and refractory period T_{ref} can be computed as [Jug et al., 2012]

$$\rho_{\text{out}} = \left(T_{\text{ref}} + \frac{\tau}{\Gamma}\sqrt{\frac{\pi}{2}} \cdot \int_{V_{\text{reset}}+k\gamma\Gamma}^{V_{\text{thresh}}+k\gamma\Gamma} \exp\left[\frac{(u-\Upsilon)^2}{2\Gamma^2}\right] \cdot \left[1+\text{erf}\left(\frac{u-\Upsilon}{\Gamma\sqrt{2}}\right)\right] du \right)^{-1} . \tag{2.28}$$

Using this approximation of firing rates allows a direct translation between the analogue activation probabilities required for CD training and the resulting firing rates of a spiking neuron with those weights. During training of the spiking DBN, the Siegert approximation is used as the nonlinearity of the neuron instead of a sigmoidal function. The predicted rate ρ_{out} in (Eq. 2.28) can be converted into a probability by normalizing with the maximum firing rate $1/T_{\text{ref}}$. This allows sampling the activation probabilities, as is done in standard CD learning with continuous-valued units. Specifically, the weight update in CD for spiking networks computes the data- and model-driven activities of the visible and hidden layer using the Siegert approximation, and then computes the weight update as usual in RBM training. Let V_{data} be the activity of the visible units driven by the input data (or activity of the hidden layer below). Then the data-driven activity of the hidden layer, given the full weight matrix W connecting the visible and hidden layer, is

$$H_{\text{data}} = \rho_{\text{out}}(V_{\text{data}}, W) \cdot T_{\text{ref}}$$

The model-driven activity of the visible and hidden layers, obtained via Gibbs sampling, is then given as

$$V_{\text{model}} = \rho_{\text{out}}(H_{\text{data}}, W^T) \cdot T_{\text{ref}}, \qquad H_{\text{model}} = \rho_{\text{out}}(V_{\text{model}}, W^T) \cdot T_{\text{ref}}$$

and the weight update Δw is

$$\Delta w = \alpha \cdot (H_{\text{data}}^T V_{\text{data}} - H_{\text{model}}^T V_{\text{model}}), \tag{2.29}$$

where α is the learning rate. After training, the parameters and weights are kept unchanged, but instead of sampling every time step, the units generate Poisson spike trains with rates computed by the Siegert formula (Eq. (2.28)). In O'Connor et al. [2013] it has been shown that this results in equivalent spiking implementations of RBMs and DBNs, which perform similarly to conventional networks with the same architecture. Neftci et al. [2014] recently proposed an online event-based implementation of the CD

learning rule for training a spike-based DBNs that may in principle utilise neuromorphic platforms. Chapter 5 will present an investigation on the robustness and scalability of spike-based DBNs to noise and reduced bit precision of neuro-inspired hardware platforms.

2.5 Summary

This chapter aimed to provide a biological background on neurons, synapses and to review a number of the available mathematical models suitable for large-scale simulations [Izhikevich, 2004]. The models presented on this chapter, however, represent only a small subset of the models available in the literature. More detailed models exist, that simulate biology at different levels of abstractions, and may be used depending on the scientific question being investigated [Sejnowski, 2003]. In addition, a number of frameworks were presented that are scalable and suitable for computations with spiking neural networks.

This large variety of models of neurons and synapses indicate that for the very active field of computational neuroscience and for the computational frameworks utilising SNNs, researchers would benefit from a programmable platform capable of simulating different and newly discovered models of neurons, synapses and plasticity rules within the same simulation. The next chapter reviews the software and hardware tools designed to simulate SNNs.

Chapter 3

Tools and Platforms for Simulating Spiking Neural Networks

3.1 Introduction

This chapter presents the different approaches for simulating large-scale spiking neural networks (SNNs) and discusses their advantages and disadvantages; every platform imposes different trade-offs between scalability, reconfigurability, and energy consumption characteristics. The first part introduces domain-specific software simulators, that are widely used by the neuromorphic research community, and then proceeds to notable simulations on supercomputers. The second part focuses on dedicated hardware architectures from off-the-shelf hardware platforms such as general-purpose graphical processing units (GP-GPUs) and field programmable gate arrays (FPGAs) to neuromorphic hardware implementations, and concludes with the SpiNNaker platform.

3.2 Software Simulators

Many scientific groups have developed their own optimised spiking neural network simulators to fit their needs. One disadvantage of developing custom simulators is that it becomes difficult for other researchers to replicate results, or extend a previously published work. Luckily over the past years a number of neural-oriented simulators have been developed to address these issues. The simulators presented here focus on simulating single-compartment neuron models, also known as point-neurons. Brette et al. [2007] divided spiking neural network simulators into three categories based on the method used to update the membrane potential of the simulated neurons. They can

be either synchronous (clock-driven) where all neurons are updated at a fixed time-step, asynchronous (event-driven) in which neurons are updated only when they receive a spike, or hybrid where the internal state of the neurons gets updated continuously according to some differential equations while spikes are expressed as events. In general, simulators using synchronous algorithms are easier to implement, however, the generated spike times are bound to specified discrete time grid which might have an impact on the precision of the results. Another factor that affects the precision of the results is that threshold conditions are checked only at the ticks of the clock. Asynchronous simulators are more complex to implement but have the advantage of simulation speed-up since there are no unnecessary calculations and spike times are precise and not bounded to a time grid. Even though spiking neural network simulators are developed in different programming languages, the majority of them offer a Python interface to them. Python is rapidly becoming the high-level language of choice for the field of computational neuroscience mainly due to the publicly available packages for plotting results (PyLab/Matplotlib[1]), efficient array data structures (NumPy[2]), and analysing neuroscience data (NeuroTools[3]).

3.2.1 NEST

NEST (NEural Simulation Tool) [Gewaltig and Diesmann, 2007], now part of the Human Brain Project [Markram et al., 2011], is a publicly available simulator that supports neural models with single or a small number of compartments. At the same time it supports heterogeneous populations of neurons and synapse models targeting simulations with more than 10^4 neurons and 10^7 to 10^9 synapses. NEST simulations can be constructed using two basic element types: nodes and connections; nodes can be either neurons or devices used to stimulate or record from neurons, connections allow nodes to link with each other. Each connection can have a user-defined delay and weight, which can be either static or dynamic, for example using the STDP [Morrison et al., 2008] rule. NEST is implemented in C++ using a global time-driven algorithm for updating neuron states but the spike-times are not constrained to the discrete time grid. NEST offers two front-ends, one utilising the native simulation language SLI and the other one uses a Python interface, named PyNEST [Eppler et al., 2009], enabling users to run simulations, analyse and plot results within the same environment. NEST

[1] http://matplotlib.sourceforge.net/
[2] http://www.scipy.org/NumPy
[3] http://www.neuralensemble.org/NeuroTools/

is capable of running on a large range of architectures, from single-core to multi-core desktop computers and supercomputers. It offers parallelisation by means of POSIX threads [Lewis and Berg, 1998] on multi-core processors and message passing interface (MPI) [Forum, 1994] for computer clusters, while ensuring reproducibility of the results regardless the number of machines or cores. The user provides a serial NEST script written in either SLI or PyNEST and the parallelisation takes places automatically. One disadvantage of NEST is that it does not allow the user to modify or add new models after the compilation process.

3.2.2 NEURON

NEURON [Hines and Carnevale, 1997], also part of Human Brain Project [Markram et al., 2011], is a simulation environment oriented for simulating detailed models of single neurons or networks of spiking neurons. One of the advantages of NEURON is that it allows users to investigate neuroscience related questions, abstracting the low-level mathematical or computational issues involved. Users can extend or modify NEURON's functionality using the NMODL language, while a graphical user interface (GUI) is available for constructing models, running simulations and analysing the results. NEURON supports three kinds of parallelism: run multiple simulations distributed over multiple processors, distributed network models with gap junctions and distributed models of individual cells. Finally, more than 1,450 scientific articles and books have reported using NEURON[4].

3.2.3 Brian

Brian [Goodman and Brette, 2009] is a spiking neural network simulator written entirely in Python enabling users to develop new models as rapidly and flexible as possible. Users can easily define arbitrary neuron, synapse and plasticity models as a set of differential equations, making it easier for researchers to share their models. Brian is a synchronous (clock-driven) simulator where all events take place on a fixed time grid, and it utilises vectorised computations during the simulations and construction of models, making it only two to four times slower than pure C code [Goodman and Brette, 2008]. Brian does not support the utilisation of parallel environments, though independent simulations may run in parallel.

[4] http://www.neuron.yale.edu/neuron/what_is_neuron#userbase

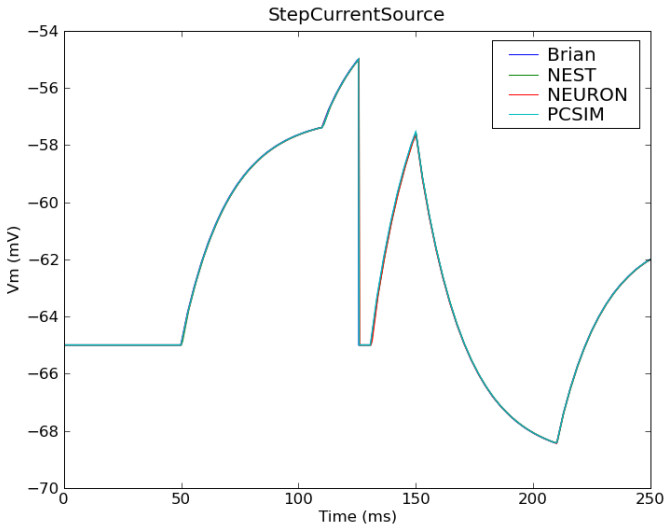


Figure 3.1: An example of using PyNN: injecting time-varying current into an LIF neuron.

3.2.4 PyNN

PyNN [Davison et al., 2009] is a common interface written in Python to multiple spiking neural network simulators. Since each simulator uses its own programming or configuration language it is time consuming and impractical to write multiple scripts of the same experiment to validate the results of the neural network. This also increases the difficulty to communicate and reproduce results between different research labs. With PyNN a researcher can write a script describing a neural network in a high-level abstraction language, using populations of neurons and projections between them, and cross-check the results on multiple simulators (e.g. NEST, Brian, NEURON, e.t.c.), as seen in Figure 3.1. Moreover, the same PyNN description can be executed on various neuromorphic platforms [Brüderle et al., 2011; Galluppi et al., 2012d] thus abstracting the hardware details from the user. Finally, PyNN uses a standard way to retrieve simulation results and since it is written in Python it is very easy to analyse and visualise results within the same environment.

3.2.5 Nengo

Nengo [Stewart et al., 2009] is not a general purpose spiking neural network simulator. It has been designed to make use of the NEF [Eliasmith and Anderson, 2004], and provides methods to define groups of neurons, form connections between them and

calculate the synaptic weights to perform the desired computations using NEF behind the scenes. Nengo is written in Java and supports a variety of spiking neuron models, and exposes a scripting interface in Python allowing users to create new models, inspect and modify neural parameters, control simulations and analyse results. Nengo offers a model repository from models presented in publications, making it easier to share knowledge between researchers. Spaun [Eliasmith et al., 2012] was implemented on Nengo and executed on high-performance clusters, occupying 24 GB of RAM, while simulations took approximately 2.5 hours for 1 second of simulated time. Galluppi et al. [2014a] demonstrated that it is possible to run Nengo models in real-time on a neuromophic robotic platform while interfacing biologically inspired event-based vision sensors.

3.3 Hardware Platforms

3.3.1 Supercomputers

Markram [2006], as part of the BlueBrain project[5], aimed to simulate a biologically plausible model of a cortical column (Blue Column), consisting of 10,000 multi-compartment models of neurons with 50×10^6 synapses. Markram argues that a detailed simulation of all neuron membrane conductances is necessary to produce a valid model of the cortex. The simulations will be executed on an IBM Blue Gene/L supercomputer which has 131,072 CPUs and 32 TB of memory. The simulation language is NEURON, which was extended to include the MPI protocol to allow communications between the nodes.

Ananthanarayanan et al. [2009], as part of the IBM/DARPA Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) project, followed a simpler approach by utilising point-neurons for their experimentation. They developed their own massively parallel simulator, named C2, which ran on a LLNL Dawn Blue Gene/P supercomputer with 147,456 CPUs and 144 TB of main memory. They simulated 1.617×10^9 neurons with 0.887×10^{13} synapses, exceeding the scale of cat cortex (or 4.5% of the human cerebral cortex). The simulation ran $643\times$ slower than real-time with 40% of the time spent on MPI communications. Based on their findings they extrapolated that in 2018 a human-brain simulation could be achievable by a supercomputer with 4 PB of memory and running at >1 EFlop/s.

[5] http://bluebrain.epfl.ch

Recently, Wong et al. [2013] simulated 53×10^{10} neurons and 1.37×10^{14} synapses on an IBM Sequoia - BlueGene/Q supercomputer with 1,572,864 CPUs and 1,572 TB of main memory. They utilised their optimised multi-threaded, massively parallel, scalable simulator named Compass [Preissl et al., 2012]. The simulation ran $1542\times$ slower than real-time, while the biggest cost reported was sending and receiving spikes via MPI messaging. The power requirements of the simulation were not reported, however the Top500 [top] list states that the peak power dissipated by the IBM Sequoia - BlueGene/Q is 7,890 kW.

Supercomputers offer great model flexibility but at the price of their large power demands and communication overheads [Anghel et al., 2014] which prohibit the real-time simulations of SNNs.

3.3.2 Graphics Processing Units

Graphics processing units (GPUs) contain a large number of processing units capable of executing a single instruction stream on multiple data simultaneously [Fatahalian and Houston, 2008]. Some of the advantages of GPUs is that they are inexpensive, available in most recent computers and can be programmed with standard development kits. GPUs have been used to accelerate ANNs with some examples including a real-time image segmentation and scene labeling [Farabet et al., 2013] or to speed-up the training process by a factor of approximately 10 to 60, compared to a compiler-optimised CPU version [Cireşan et al., 2011].

Fidjeland et al. [2009] presented the NeMo SNN simulator which ran on an Nvidia Tesla C1060 GPU. NeMo is capable of simulating 40×10^3 Izhikevich neurons with 10^3 synapses per neuron, and a mean firing rate of 10 Hz. This simulator supports axonal delays, different network topologies and can deliver up to 400×10^6 synaptic events (SE) per second. Fidjeland and Shanahan [2010] extended NeMo to include STDP plasticity and optimised its performance to deliver up to 550×10^6 SE per second. NeMo offers a programming interface similar to PyNN to allow users to run their experiments without any background on GPUs. Nageswaran et al. [2009] demonstrated their SNN simulator on an Nvidia GTX-280. They were able to simulate up to 100×10^3 Izhikevich neurons with 50×10^6 synapses and a mean firing rate of 7 Hz. In addition, axonal delays and STDP is supported, while they reported that the simulations ran $26\times$ faster than the single-threaded simulator written in C on a CPU version. GPUs are excellent platforms for accelerating algorithms that have high computation to communication ratio [Brette and Goodman, 2012]. However, this is not true for SNN simulations which require

frequent memory access and they are communication bound [Moore et al., 2012]. Both Fidjeland et al. [2009] and Nageswaran et al. [2009] reported that the limiting factor of their work was the memory bandwidth. Programming a GPU becomes a non trivial task as it requires a careful mapping of the data structures in memory to achieve the available maximum bandwidth [Brette and Goodman, 2012; Fatahalian and Houston, 2008]. Finally, the power requirements of GPUs [Jiao et al., 2010], which are in the order of several hundreds of watts, might be an issue for some applications for example robotic platforms.

3.3.3 Field Programmable Gate Arrays

Custom hardware implementations of SNNs may significantly outperform general-purpose processors. Some of the advantages of FPGAs are that they are reconfigurable, they provide fast performance and they reduce the costs and effort of fabricating application specific integrated circuits (ASICs). In addition, the design code can be shared between researchers to allow collaborations and sharing of knowledge.

Cassidy et al. [2011] proposed an architecture of neural computational arrays, where instead of physically simulating neurons on hardware they multiplex multiple neurons into one physical neuron. Their implementation uses LIF neurons with delta-current synapses, however the neural model can be easily modified to accommodate any arbitrary point-neuron model (e.g. Izhikevich neuron). The AER protocol is used for communicating the spikes, while their implementation targets real or accelerated execution time. The limiting factor of their system was reported to be the size of the external RAM and the communications bandwidth. Their analytical results were verified on a Xilinx Virtex 6 SX475 FPGA clocked at 200 MHz with a 36 Mb QDR SRAM were the mapping and synapses are stored. Neither the topology of the neural network nor the number of bits used for the synapses were reported.

To address the communication bandwidth issues that arise with large-scale, real-time simulations of SNNs Moore et al. [2012] have engineered a scalable multi-FPGA architecture, named BlueHive (Figure 3.2), that follows a communication-centric approach. The building block of their system is an Altera Stratix IV 230 FPGA with custom PCBs to break out serial links to pluggable SATA channels thus providing a 72 Gb/s bandwidth per FPGA. Each FPGA, clocked at 200 MHz, can simulate up to 64×10^3 Izhikevich neurons in real-time with 64×10^6 delta-current synapses while supporting axonal delays. The high-speed communication interfaces are capable of delivering inter-board spikes in under a millisecond, 50 ns per hop. Neural network parameters

are held in the external memory thus supporting different topologies without having to reconfigure the FPGAs. BlueHive is scalable to up to 64 FPGAs, while a 4-FPGA system with 256×10^3 neurons and 256×10^6 synapses has already been demonstrated. Performance-wise, BlueHive ran $162\times$ faster compared to a single-threaded SNN simulator written in C and executed on an Intel Xeon X5560 2.80 GHz server with 48 GB of RAM. However, power requirements were not reported.

Neil and Liu [2014] introduced Minitaur, an event-driven implementation for accelerating SNNs simulations, with a particular interest in the applications of spiking DBNs. Minitaur uses event-driven LIF neurons with delta-current synapses, where the membrane is updated only at the arrival of input spikes thus reducing the computational costs. In addition, the computation speed depends on the network activity and not the number of neurons. Minitaur was implemented on a low-cost Xilinx Spartan 6 LX150 with 128 MB of DDR2 RAM and 128 KB of flash memory, clocked at 75 MHz. It achieved a peak performance of 18.73 Million synaptic events (SE) per second while dissipating 1.5 W. The maximum number of neurons supported by the platform is 65,536 with 16.78 million synapses.

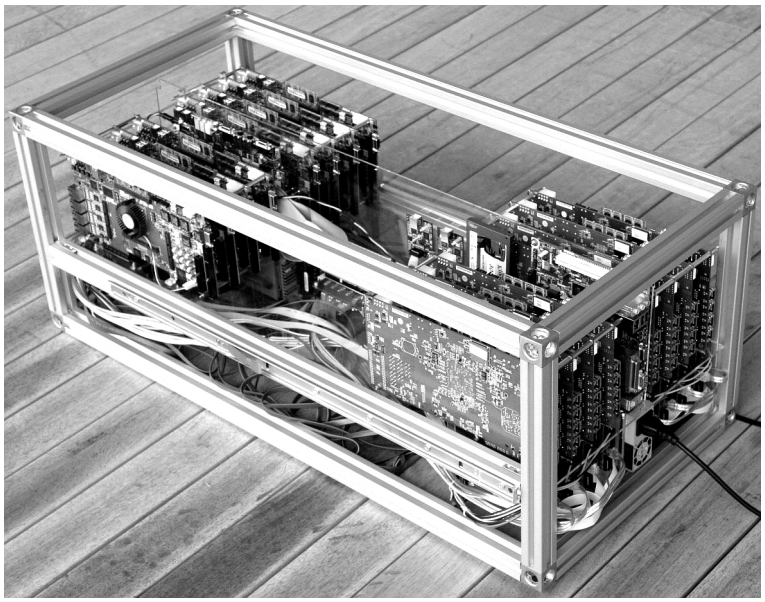


Figure 3.2: A BlueHive rack with 16 FPGA boards [Moore et al., 2012].

While FPGAs have the advantage of being reconfigurable they suffer from their high power and area requirements compared to ASICs. However, this gap in power, area and performance tends to get smaller with every new technological iteration [Kuon and Rose, 2007], while a successful design on an FPGA can always be turned into a

higher-performance ASIC in the future. Lastly, depending on the background of the researcher, implementing new models or extending the functionality of a design might not be a trivial task.

3.3.4 Neuromorphic Hardware

Neuromorphic engineering, a concept originally developed by [Mead, 1990], aims to provide real-time simulations of spiking neural networks in an energy efficient manner. Depending on how neurons, synapses and spike transmission are implemented neuromorphic systems can be categorised as either analogue, digital, or mixed-mode analogue/digital VLSI circuits. Analogue implementations exploit the sub-threshold transistor dynamics [Indiveri et al., 2011] and are more energy-efficient while requiring less area than their digital counterparts [Joubert et al., 2012]. However, the behaviour of analogue circuits is largely determined during the fabrication process due to transistor mismatch [Indiveri et al., 2011; Pedram and Nazarian, 2006; Linares-Barranco et al., 2003], and also analogue neuromorphic circuits suffer from scaling limitations due to the large capacitances required by the models [Joubert et al., 2012]. The majority of mixed-mode analogue/digital neuromorphic platforms use digital packet-based technology to communicate spikes as AER events. This enables reconfigurable connectivity patterns while the time of spikes is expressed implicitly since typically a spike reaches its destination in less than a millisecond, thus fulfilling the real-time requirement. Neuromorphic systems suffer from model flexibility, since neurons and synapses are fabricated directly on hardware with only a small subset of parameters available to the researcher. In addition, it requires effort to design and they are costly to fabricate. This subsection aims to provide a brief review of some of the large-scale neuromorphic systems, as well as, biologically inspired neuromorphic sensors currently available.

3.3.4.1 SyNAPSE/TrueNorth

IBM researchers, in collaboration with a number of universities, have been working on a project supported by the Defence Advanced Research Programs Agency (DARPA) named Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE). The SyNAPSE project aims to develop a neuromorphic machine capable of simulating neurons in real-time with 10^{14} synapses while consuming 1 kW of energy [Sawada and Modha, 2013]. The SyNAPSE project can be divided into two parts, hardware and

software.

The software side consists of Compass [Preissl et al., 2012] a multi-threaded, massively parallel, scalable simulator that has a one-to-one equivalence to the functionally of TrueNorth [Merolla et al., 2014b]. This one-to-one correspondence enabled the development of an abstract neural programming model named Corelet [Amir et al., 2013]. With Corelet users can compose and configure cognitive systems that may be executed on Compass or on the TrueNorth architecture thus abstracting the hardware details and allowing code re-usability.

The hardware side consists of the TrueNorth architecture and chip. The basic building block of a TrueNorth chip is the neurosynaptic core, which consists of 256 neurons, 256 axons, and 256×256 binary synapses implemented as a crossbar memory. Axons can be set to one of four types (excitatory or inhibitory), and each post-synaptic neuron can individually assign a programmable signed integer value for that axon type. Neurons are implemented directly on hardware and they are multiplexed in time, 256 neurons can be evaluated in a single time-step. They are updated at discrete time-steps, default value 1 ms. Neural parameters and states are stored locally in a 256×410 bit SRAM, where each row stores all relevant information for each neuron. The neuron model implemented is the LIF neuron [Cassidy et al., 2013] with delta-current synapses. Spikes are transmitted as destination-based AER packets and each neurosynaptic core has a router whose purpose is to route packets between adjacent cores to the destination cores with programmable axonal delays. A single TrueNorth chip comprises 4,096 neurosynaptic cores and is capable of simulating up to 10^6 neurons with 256×10^6 configurable synapses in real-time. TrueNorth is a hybrid asynchronous-synchronous chip. The communications and control circuits are asynchronous, while the synchronous computations are clocked by the asynchronous control, thus minimising the active power.

TrueNorth was demonstrated on a multiobject detection and classification task in real-time while consuming 63 mW. Compass running the identical task on an Intel Core i7 CPU 950 with 4 cores and 8 threads, clocked at 3.07 GHz was 100 to 200× slower than real-time and consumed 100,000 to 300,000 × more energy per synaptic event (SE) [Merolla et al., 2014b]. Finally, TrueNorth is capable of delivering 46 billion synaptic operations per second (SOPS) per Watt in real-time, while consuming 26 pJ per SE, and 2.3 pJ to route a spike per hop.

3.3.4.2 HICANN

Brain-inspired multiscale computation in neuromorphic hybrid systems (BrainScaleS), now part of the HBP, introduced the High Input Count Analog Neural Network (HICANN) [Schemmel et al., 2010] chip, which is a mixed-signal neuromorphic architecture. The building block of HICANN is the analogue network core (ANC) which is configurable crossbar circuit that can have 8 neurons with 16×10^3 pre-synaptic inputs per neuron, or 512 neurons with 256 inputs per neuron. The implemented neuron model is the adaptive exponential integrate-and-fire model [Brette and Gerstner, 2005], and the synapses are represented as current generated by a 4-bit multiplying DAC [Linares-Barranco et al., 2003]. The precision of the synapses can be increased to 8-bits but at the cost of a twofold reduction of their number. In addition, the synapses offer STDP plasticity, which is controlled by a programmable digital circuit. Neurons in HICANN run in accelerated biological time that ranges from 10^3 up to 10^5 compared to biological real-time. In order to overcome the communication overheads they implemented a wafer-scale scheme with digital packet-switched routers for spike transmission [Schemmel et al., 2008] providing an intra-wafer bandwidth of up to 2 Gb/s. A 20 cm wafer can fit 352 HICANN chips allowing simulations of 4×10^7 synapses and up to 180×10^3 neurons. Larger systems can be created by combining several wafers together with an inter-wafer bandwidth of 44 Gigaevents/s and a maximum latency of 4 ns per hop.

3.3.4.3 Neurogrid

Neurogrid [neu, 2006] is a hybrid analogue/digital neuromorphic architecture. A neurogrid board consists of 16 neurocores connected in a binary tree, a Xilinx Spartan-3E FPGA and 8 Cypress 4 MB SRAMs used for axon-branching. Each Neurocore chip is capable of emulating up to 2^{16} two-compartment neurons in real-time, while consuming less than 1 W. Spikes are propagated as AER packets using an event-based digital communication network providing a bandwidth of up 1.17 Gwords/s with less that 1 µs latency along the longest path. On a neural network simulation that comprised 10^6 neurons with 8×10^9 synapses Neurogrid consumed 3.1 W, with 941 pJ per synaptic event [Benjamin et al., 2014].

3.3.4.4 HiAER-IFAT

HiAER-IFAT [Yu et al., 2012] is a mixed-signal VLSI with 2^{16} two-compartment LIF neurons, each with four time-multiplexed conductance-based synapses. Neurons and

synapses are emulated using sub-threshold transistor dynamics with a programmable set of parameters. In order to allow arbitrary connectivity between neurons, spikes are communicated as AER events using five Xilinx Spartan-6 FPGA chips, each with four 2 Gb DDR3 RAM for storing the synaptic look-up tables. HiAER-IFAT provides a total throughput of up to 3.6×10^7 SE/s, with 5 Mevents/s of maximum input rate per neuron, while consuming 252 μW (50 pJ/spike). The spike latencies reported were below 450 μs [Park et al., 2012].

3.3.4.5 Sensors

A dynamic vision sensor (DVS) consists of 128×128 pixels where each pixel responds asynchronously to relative changes in intensity. Contrary to conventional frame-based vision sensors that wastefully transmit entire images at fixed frame rates, only the local pixel-level changes caused by moving in a scene are transmitted. The sensor outputs an asynchronous stream of pixel-addresses, encoded as AER packets, at exactly the time they occur. This results in very small latencies, in the order of 15 μs [Lichtsteiner et al., 2008] or 3.6 μs [Leñero-Bardallo et al., 2011], low-power consumption (23 mW) and an increase in the dynamic range (>120 dB) due to the local processing.

Liu et al. [2010] developed an event-based binaural silicon cochlea that emulates the hair cells, ganglion cells and the basilar membrane and is aimed for spatial audition and auditory scene analysis. Spikes from each ear are encoded as AER packets and transmitted asynchronously with a maximum bandwidth of 10 Mevents/s and a latency of 2 μs. The reported dynamic range is 36 dB, while the peak power dissipation for the digital circuits is 25 mW and 33 mW for the analogue ones.

3.4 The SpiNNaker Architecture

The Spiking Neural Network Architecture, or SpiNNaker, is a biologically inspired, massively-parallel, scalable computing architecture designed by the Advanced Processor Technologies (APT) group at the University of Manchester. SpiNNaker has been optimised to simulate very large-scale spiking neural networks in real-time [Furber et al., 2014]. Each SpiNNaker chip comprises 18 identical low-power fully programmable cores that allow the use of dynamic, arbitrary and heterogeneous models in simulation. The key innovation of the SpiNNaker architecture is its communications infrastructure designed to cope with very small frequent packets of spiking events, scalable up to a billion neurons [Navaridas et al., 2010]. Spikes are transmitted as AER packets, with

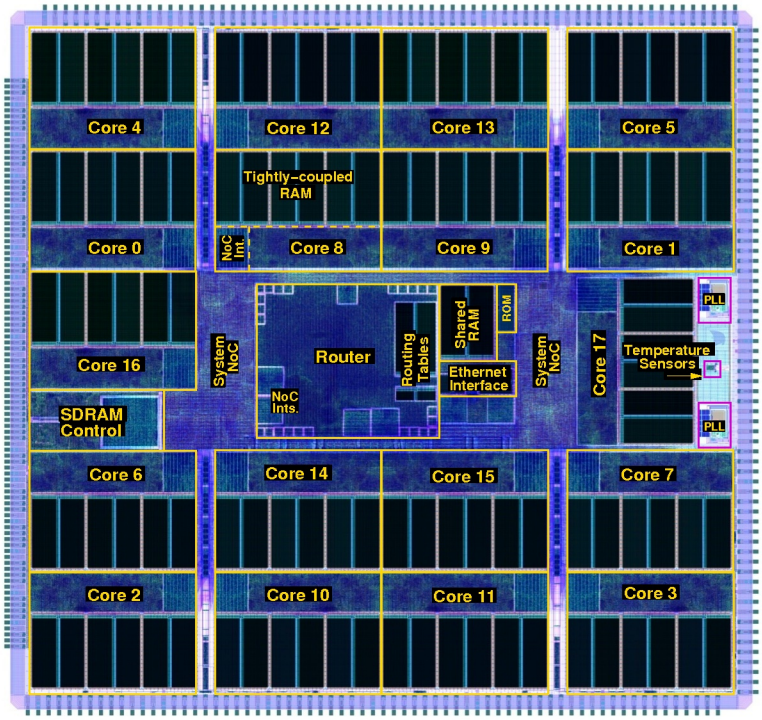
their time expressed implicitly as the communications fabric can deliver a packet in much less than a millisecond, fulfilling the requirement for real-time neural processing. By connecting SpiNNaker chips and boards together, large machines can be formed, permitting simulations to scale-up seamlessly; the final SpiNNaker machine aims to simulate a billion neurons with trillions of synapses in real-time.

The SpiNNaker architecture combines some of the advantages of conventional computing platforms and neuromorphic hardware. While not achieving the energy efficiency of dedicated neuromorphic hardware its programmability allows the development of, and experimentation with, new neuron, synapse and plasticity [Galluppi et al., 2014b] models, while its communications infrastructure overcomes the limitations of general-purpose supercomputers enabling real-time simulations.

3.4.1 Hardware Architecture

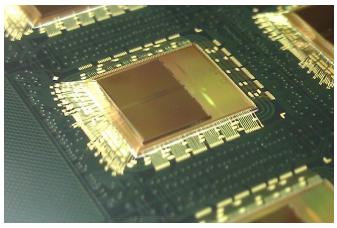
A SpiNNaker chip, shown in Figures 3.3(a) & (b), comprises 18 identical ARM968 cores each with its own local tightly-coupled memory (TCM) for storing data (64 kB) and instructions (32 kB). All cores have access to a shared off-die 128 MB low-power dual-data-rate (DDR) SDRAM (Figure 3.3(b)) through a self-timed system network-on-chip (NoC) where the relevant synaptic information is stored (Figure 3.4). In addition, every ARM9 core has a communications controller responsible for generating and receiving neural spikes to and from the router through a self-timed communications NoC, a direct memory access (DMA) controller for reading and writing to the shared off-die SDRAM, and two timer peripherals, one of which is used to generate interrupt signals periodically to solve the neural equations. Out of the 18 ARM9 cores, 16 may be used for neural applications known as application cores, 1 is used for monitoring and administration purposes also known as monitor core, and 1 is spare for fault tolerant purposes [Furber et al., 2013]. Jin et al. [2008] demonstrated that each ARM9 core is capable of simulating, in real-time, 1,024 Izhikevich neurons, each receiving 1,024 delta-current connections and firing with a mean firing rate of 10 Hz. This however was achieved with optimised assembly code and in practice the number of neurons and synapses that can be simulated by a single core depend on the computational complexity of the models and, in particular, by the number of input connections, their firing rates, and the complexity of the synapse model.

Spikes are transmitted as 40 or 72 bit multicast (MC) packets implementing the source-based Address-Event Representation (AER) [Mahowald, 1994] scheme, where the information transmitted is the address of the firing neuron. Each packet consists



(a)

(b)



(c)

Figure 3.3: System overview of SpiNNaker: (a) The SpiNNaker chip die. (b) Packaged production SpiNNaker chip. (c) A SpiNNaker chip and stacked SDRAM.

of an 8 bit packet header, a 32 bit routing key identifying the neuron that fired and an optional 32 bit payload which is not normally used for neural applications [Wu and Furber, 2010].

At the center of the SpiNNaker chip lies a packet-switched MC router [Wu and Furber, 2010] responsible for communicating the spikes to the local ARM9 cores or to neighbouring chips through 6 asynchronous bi-directional links namely: East,

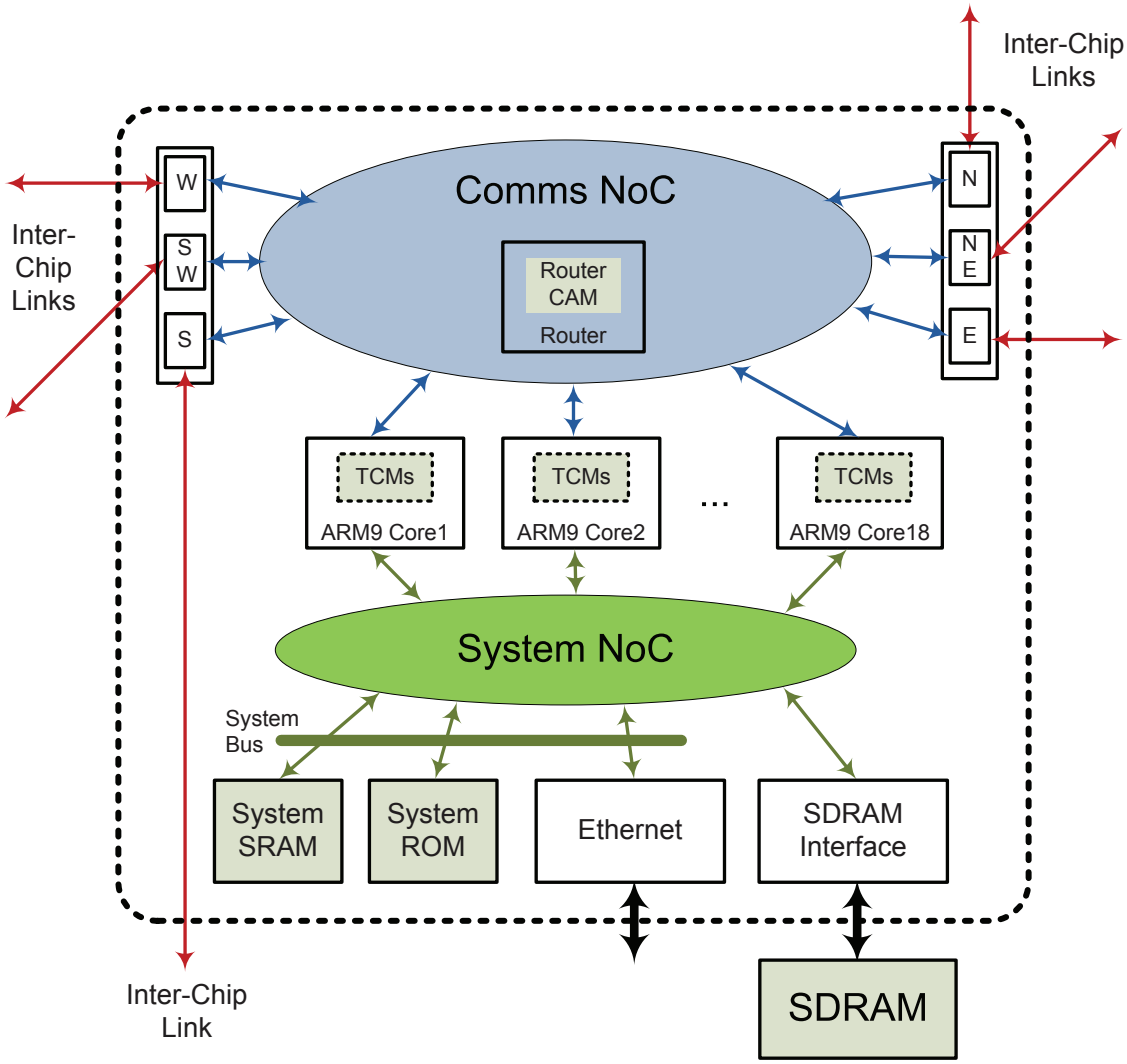


Figure 3.4: Simplified Block diagram of a SpiNNaker chip [Patterson, 2012].

North-East, North, West, South-West, South as seen in Figure 3.4. The bi-directional links support an aggregated spiking bandwidth of 1.5 Gb/s [Patterson et al., 2012] and arbitrary connectivity [Furber et al., 2006]. Each SpiNNaker router has 1,024 routing tables implemented using content addressable memory (CAM). Two additional features of the SpiNNaker router are the default routing and emergency routing. Default routing takes place when no CAM entry matches the key of a MC packet; the router will then route the packet to the output link opposite the input link through which it arrived thus minimising the total number of the required routing entries per node. Emergency routing takes place when a problematic link is detected either due to transient congestion or link failure. In the case of an emergency routing the router will attempt to route the MC packet via the link rotated one link clockwise from the blocked link. The

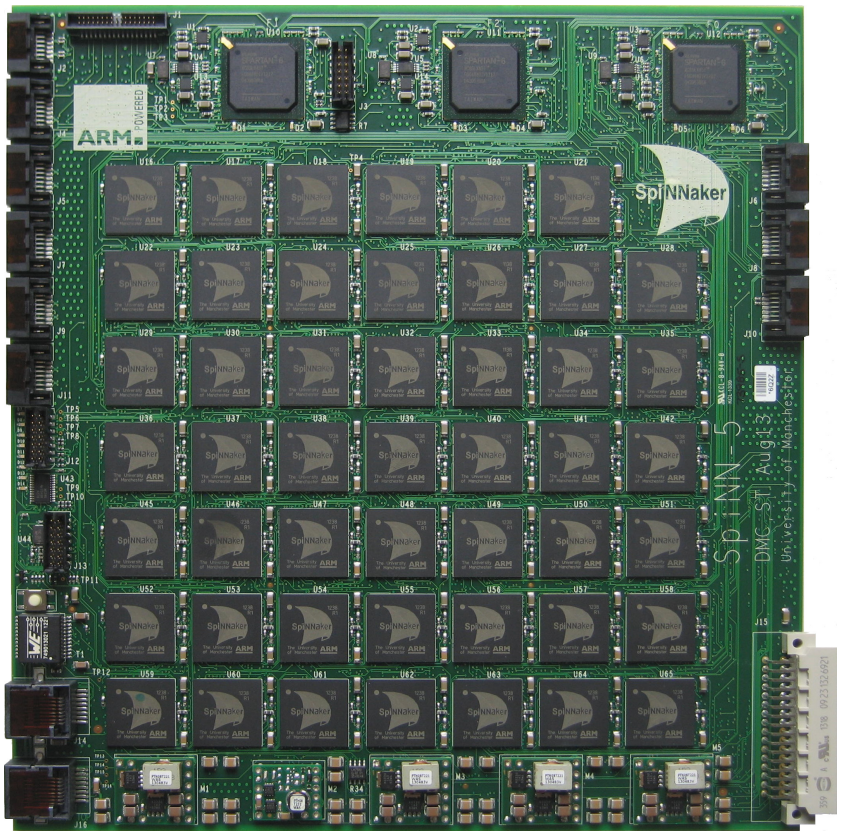


Figure 3.5: A board with 48 SpiNNaker chips (SpiNN-5).

router is capable of handling one-to-many communications efficiently, while its novel interconnection fabric allows it to cope with very large numbers of very small packets. A study by Navaridas et al. [2010] verified that the SpiNNaker routers are capable of processing packets at a rate which scales up to billion neuron simulations.

By combining multiple SpiNNaker chips together larger systems are formed. The SpiNNaker printed circuit board (PCB) with 48 chips (SpiNN-5), as seen in Figure 3.5, is the largest prototype system available to-date and is currently being used as the building block for forming larger SpiNNaker machines. The SpiNN-5 board has 864 ARM9 cores, 768 of which can be used for neural applications, while 1 core per chip is dedicated for monitoring purposes and an additional one for fault-tolerant purposes [Furber et al., 2013]. Additionally, there are 3 Xilinx Spartan-6 FPGA chips used for inter-board communication purposes through the 6 high-speed SATA links. The final SpiNNaker machine will utilize approximately 1,200 SpiNN-5 boards with estimated peak power requirements at around 75 kW [Furber et al., 2014], and it aims at simulating one billion neurons with trillions of synapses in biological real-time.

3.4.2 Software for Neural Simulations

One of the main advantages of the SpiNNaker platform is its reprogrammability. On the application level this means that each ARM9 core may execute different neuron and synapse models, plasticity rules or any type of event-driven software in the same simulation. On the communications level, the SpiNNaker routers are capable of allowing any arbitrary network topology by programming their routing entries. This section will present the SpiNNaker system software and how spiking neural networks are mapped and executed on a SpiNNaker system.

3.4.2.1 The SpiNNaker Software

The SpiNNaker software can be divided into two parts, the software running on the chips and on the host-side, which could be any general purpose computer. Each SpiNNaker core runs an event-based Application Run-time Kernel (SARK) that has two threads which share the processor time, the scheduler and the dispatcher. The scheduler is responsible for queuing tasks based on a user-defined priority, while the dispatcher de-queues and executes them starting with the highest-priority task. Tasks with priorities set to one and above are queueable [Sharp et al., 2011] and are placed into a scheduling queue based on their priority. Zero task priorities are non-queueable and are executed directly from the scheduler, while they may pre-empt the execution of a queueable task. Finally, users are allowed to set one task to minus one priority (highest priority) making it a non-queueable, pre-eminent and will pre-empt other non-queueable and queueable tasks. This can be seen in Figure 3.6(a).

The SpiNNaker application programming interface (API) is built on top of SARK and allows users to write sequential C code to describe event-based neuron and synapse models by assigning callback functions that respond to particular system events. Some of the commonly used events for neural applications include:

- **Timer Event:** A user-defined periodic event, usually set to every millisecond, triggered by the hardware timer; this is used to solve the neural equations and update the synaptic currents. If the membrane potential of a neuron crosses its predefined threshold value a spike (MC packet) is generated with the address of the neuron that fired and send to the router.

- **Packet Received Event:** An event is triggered every time a core receives a spike (MC packet). It initiates a DMA transfer to fetch the pre-synaptic information

from the SDRAM to the local TCM memory. This DMA operation is autonomous, the ARM core may handle pending events or enter into a power-saving sleep mode.

- **DMA Done Event:** This event is generated by the DMA controller to inform the core that a DMA transfer has been completed. Each synaptic weight and conductance delay gets updated.

This event-driven software model can be observed in Figure 3.6(b). If there are no pending tasks the cores enter a low-power sleep mode.

3.4.2.2 Mapping Spiking Neural Networks on SpiNNaker

Configuring a million-core machine designed to simulate a billion neurons with trillions of synapses, while allowing arbitrary network topologies is a non-trivial problem. To address this issue a tool named partition and configuration management (PACMAN) [Galluppi et al., 2012d] was developed. PACMAN is a set of algorithms that takes a high-level description of a neural network topology as its input and maps it on a SpiNNaker system based on the available resources.

PyNN [Davison et al., 2009] is used as a high-level neural specification language, allowing users to describe neural topologies and parameters using abstractions such as *populations* and *projections*. The advantage of using PyNN as a user interface is that the SpiNNaker platform becomes available to non-expert users by abstracting the hardware details, and that the results can be validated against software simulators such as Brian, Nest and NEURON.

In Figure 3.7 the flow chart of PACMAN's algorithms can be observed (left), and the data representations they operate on (right). The model view represents a neural network described in PyNN, it is then split into sub-populations and sub-projections based on the number of neurons and synapses a single ARM9 core can execute (PACMAN view). The next step is an intermediate stage where the sub-populations and sub-projections are mapped and routed on a physical machine instance (System view) using the information stored in the system library. The translation process uses information stored in the model library to generate the binary files and executables for the neural/synapse models and their parameters which are going to be stored in the local TCMs of the ARM9 cores; the synaptic information such as weights, delays and type of synapse that will be stored in the SDRAMs; and the routing entries to route the spikes to the appropriate

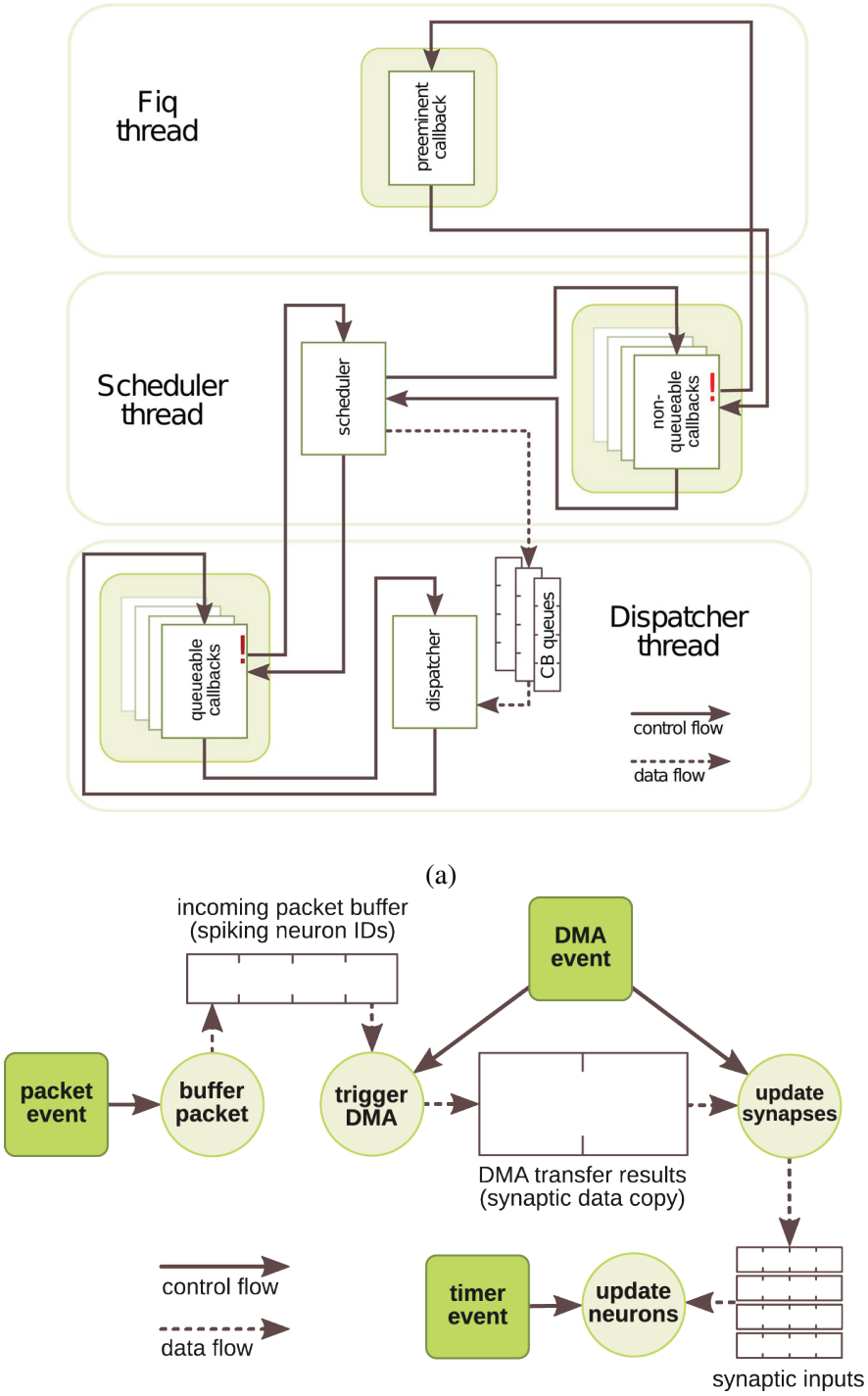


Figure 3.6: Software overview: (a) Control and data flow between the scheduler and the dispatcher. (b) Event-driven software API of SpiNNaker [Sharp et al., 2011].

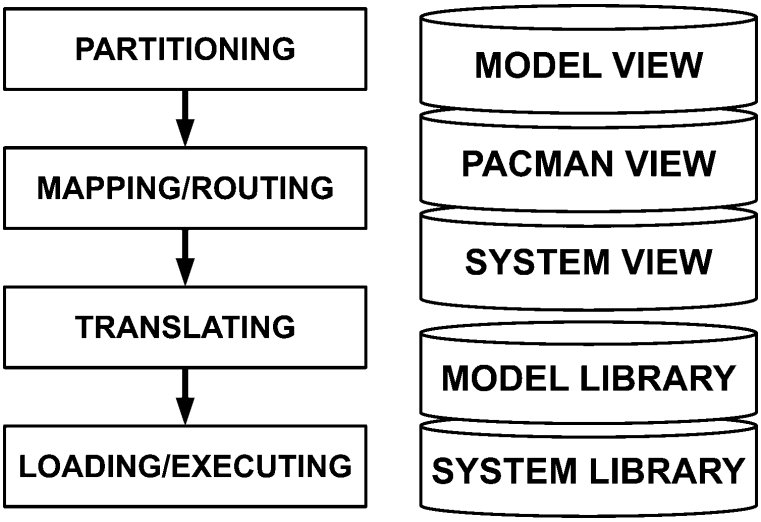


Figure 3.7: Flow chart of PACMAN's algorithms (left) and the data representations they operate on (right) [Furber et al., 2014].

destinations. Finally, the binary files and executables are loaded onto a SpiNNaker system, an example of this can be seen in Figure 3.8, and the simulation is executed.

3.4.3 Applications

This subsection highlights the flexibility of the SpiNNaker platform and provide a brief summary of some of the applications where it has been successfully utilised.

Sharp et al. [2014] simulated, in real-time, the model of the rodent barrel cortex, which consisted of 50×10^3 LIF neurons and 50×10^6 current-based exponential synapses. Galluppi et al. [2014b] by taking advantage of the programmability and massive parallelism of the SpiNNaker platform introduced a novel framework for real-time plasticity on SpiNNaker; for every neural population requiring plasticity PACMAN allocates a separate core. The advantage of this approach is that the plasticity rules are decoupled from the neuron and synapse models, while it is relatively easy to develop new learning rules. To demonstrate the latter, three plasticity rules were implemented: STDP [Morrison et al., 2008], voltage-dependent STDP [Brader et al., 2007] and the rate-based BCM rule [Bienenstock et al., 1982]. Recently, Lagorce et al. [2015] implemented a set of purely event-driven neuron and plasticity models on SpiNNaker to enable sub-millisecond simulations for applications like sound localisation which require finer time resolution than milliseconds [Knudsen, 2002]. Galluppi et al. [2012c] integrated a DVS silicon retina with SpiNNaker to implement a real-time feature-based attentional selection model. Finally, Galluppi et al. [2014a] by taking advantage of

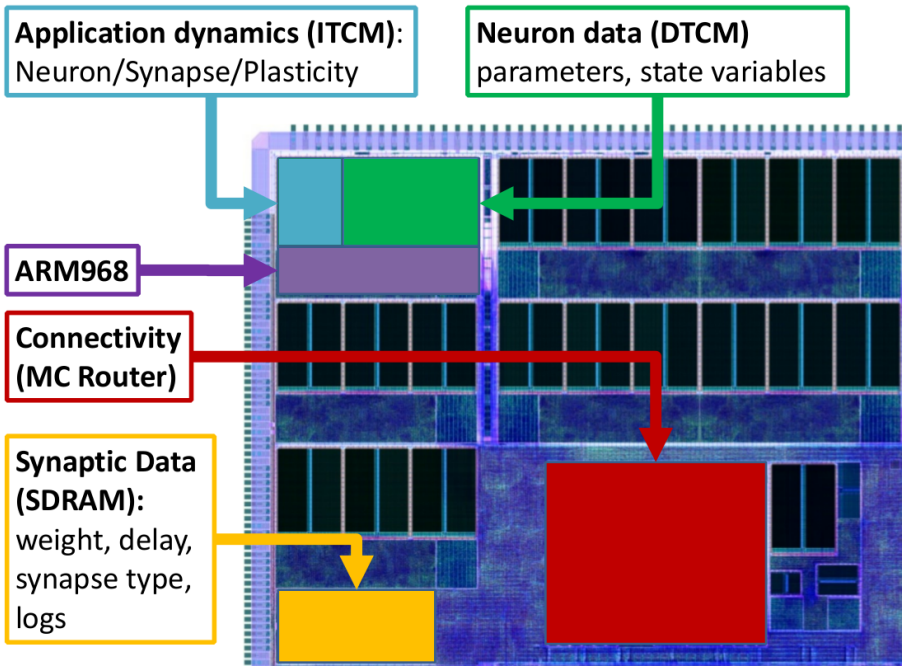


Figure 3.8: Functional blocks of a subsection of a single SpiNNaker chip [Galluppi, 2013].

the low-power operation of the SpiNNaker cores, coupled a SpiNN-5 board to a mobile robotic platform to enable simulations of SNNs that need to interact with the environment in real-time.

Apart from PyNN, PACMAN has been extended to interface SpiNNaker with the Nengo [Galluppi et al., 2012b] simulator. This enables researchers to take advantage of the NEF and create networks of spiking neurons that implement desired computations in real-time. Some applications of NEF on SpiNNaker include a model of rat hippocampus place, grid and border cells on a mobile robot [Galluppi et al., 2012a]. Galluppi et al. [2014a] demonstrated a Nengo model on a mobile robotic platform that used two silicon retinas as inputs and the task was to keep a stimulus input at a fixed distance and orientation. Mundy et al. [2015] recently optimised the NEF implementation on SpiNNaker and now requires 90% less memory, while each SpiNNaker core can simulate up to 2,000 neurons. Future plans include the real-time realisation of Spaun [Eliasmith et al., 2012] on larger SpiNNaker machines [Furber et al., 2014; Mundy et al., 2015].

3.5 Summary

This chapter aimed to provide a review of the available software and hardware platforms for simulating large-scale SNNs. General-purpose software simulators offer model flexibility and are simple to use but also unable to efficiently utilise the computational parallelism required for large-scale simulations [Morrison et al., 2005]. Similarly, supercomputers offer great parallelism but suffer from communication overheads and vast power requirements. Other off-the-shelf hardware platforms such as GPUs have the advantage that they are affordable but it is not trivial to program if one requires to exploit the maximum capacity of the available memory bandwidth [Fidjeland et al., 2009; Nageswaran et al., 2009], while their energy requirements might be issue for some applications. FPGAs are reconfigurable and cost-efficient solutions compared to fabricating an ASIC but most SNNs implementations show little model flexibility and the memory bandwidth is reported to be the limiting factor [Moore et al., 2012; Cassidy et al., 2011]. Neuromorphic approaches have proven to be an energy-efficient solution by simulating the neurons and synapses directly on hardware, and enable real- or accelerated-time simulations of SNNs by employing packet-based routing techniques for spike propagation [Merolla et al., 2014b; Schemmel et al., 2008]. However, they suffer from model specialisation; it takes extra effort and cost to fabricate an ASIC and may have a steep learning curve for new users. SpiNNaker aims to combine the advantages of conventional computers and neuromorphic hardware by utilising low-power programmable cores and scalable event-driven communications hardware.

Figure 3.9 presents a qualitative summary of the advantages and disadvantages of each platform discussed in this chapter, where for each platform the term energy efficiency refers to the contributions of the energy consumed: *a)* per synaptic event, *b)* to simulate a neuron for a single time-step, and *c)* to transmit a spike from one computational node to the next. Biological fidelity means how close to biology the neuron, synapse and spike transmission circuits are, or for the case of digital platforms if it is possible to program more detailed models (for example multi-compartment models). Communication latency refers to the time required to transmit a spike from one computational node to the next. Cost means the costs related to design, fabricate (for the case of ASICs) or purchase for the case of off-the-shelf hardware platforms such as super computers, GPUs, or FPGAs. The term scalability means if the platform is capable of simulating larger models (increasing the number of neurons/synapses) just by adding computation nodes to the system. Programmability refers to the possibility of programming new neuron, synapse, or plasticity models on the same hardware.

One of the main contributions of this thesis is presented in the next chapter; that shows a method to characterise the power requirements and communication latencies of the SpiNNaker platform, as well as, techniques to minimise overall power usage.

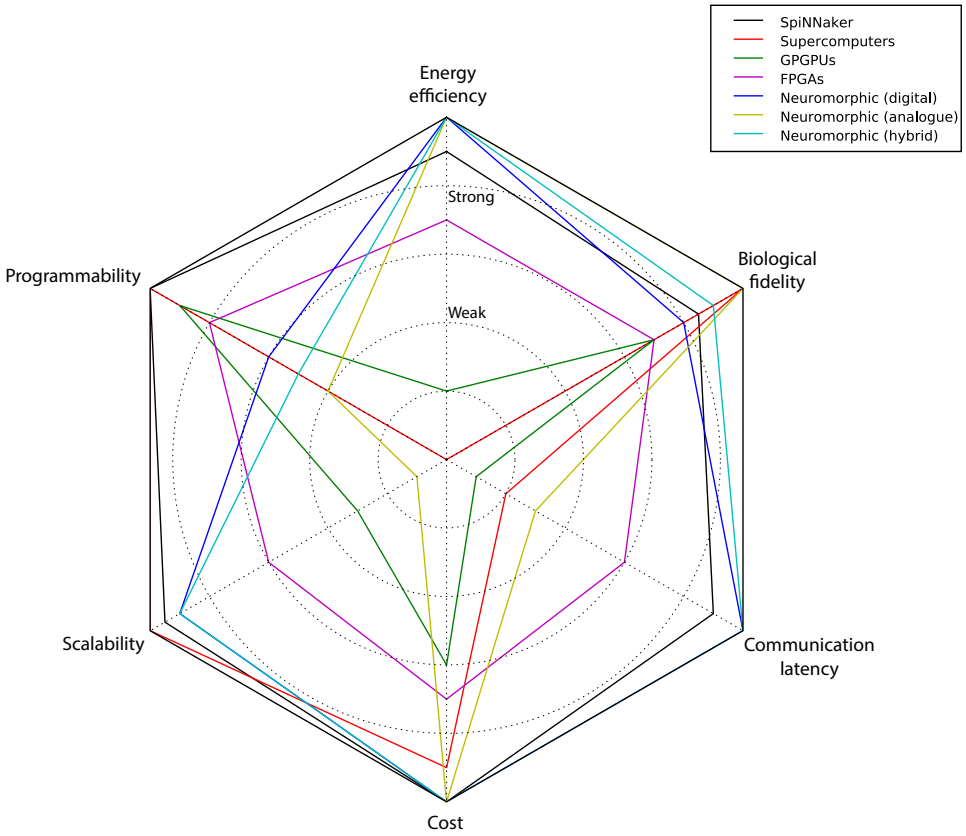


Figure 3.9: A qualitative comparison of different platforms for simulating SNNs.

Chapter 4

Power and Latency Characterisation of the SpiNNaker Platform

SpiNNaker has been designed to enable large-scale simulations of heterogeneous models of spiking neurons in biological real-time [Furber and Brown, 2009]. As described in the previous chapter, by combining several SpiNNaker chips together larger systems are formed. For the research presented in this chapter a SpiNN-4 board was used, which is a prototype of a SpiNN-5 board, and was the only board available with 48 SpiNNaker chips at the time of this research. This chapter aims to sketch the power dissipation profile of the SpiNNaker platform, characterise the spike latencies as imposed by the software overheads and communication fabric and propose new modes of operation to further reduce the dissipated power.

Section 4.1 describes the common experimental set-up for the remaining experiments. This includes a description of the neuron and synapse models used, the proposed benchmark neural networks and the rationale behind them, the parameters that monitored the status of the simulations to guarantee correctness of results, and finally the methodology employed for measuring the power dissipation during the simulations.

Section 4.2 presents, the simulation of large-scale, real-time simulations of up to a quarter of a million neurons, tens of millions of synapses, generating an activity of more than a billion synaptic events per second, which at the time of publication [Stromatias et al., 2013] was the largest recurrent real-time simulation of spiking neural networks. More importantly, this section focuses on a controlled and systematic simulation environment, using the proposed benchmark neural topologies, to obtain a more general power characterisation equation for SpiNNaker based on numbers of neurons, synapses and their firing rates and the energy required per synaptic event.

This work has been previously published by Stromatias, Galluppi, Patterson and Furber (2013) in the International Joint Conference on Neural Networks.

Section 4.2.2 presents an investigation on the intra- and inter-chip spike latencies as a function of synthetic background traffic. This section reproduces the author's contribution to the work of Lagorce, Stromatias, Galluppi, Plana, Liu, Furber and Benosman (2015) in Journal of Frontiers in Neuromorphic Engineering.

Section 4.3 presents a systematic investigation into the overall energy consumption of a SpiNN-4 board and suggests a number of optimised suspend modes to reduce this. The proposed implementation shows significant energy savings during the simulation phases, when compared to the default implementation. This work has been previously published by Stromatias, Patterson, and Furber (2014) in the International Conference on Neural Networks.

4.1 Experimental Set-up

4.1.1 Neuron and Synapse Models

Two spiking neuron models were used, the LIF and the Izhikevich (IZK) model, described in sections 2.3.1.2 & 2.3.1.3, while for the synapse model the current-based instantaneous rise and single-exponential decay model was utilised, as described in section 2.3.2.1. The full set of neural and synapse parameters used in the experiments can be found in Table 4.1, for the LIF model, and Table 4.2, for the IZK model. Values inside the brackets indicate the range of a randomly generated variables based on a uniform distribution. The random seed is kept constant throughout the experiments. The E or I subscripts represent the excitatory and inhibitory post synaptic currents (PSC). Finally, $\tau_{E/I}$ represents the decay time of the excitatory/inhibitory synaptic currents.

The SpiNNaker ARM968E-S cores do not include a floating-point unit (FPU) thus the internal states of the neuron and synapse models are computed using fixed-point arithmetic [Jin et al., 2008].

4.1.2 Benchmark Neural Network Topologies

This section describes the networks used to test the system in a controlled way for both the LIF and the Izhikevich neurons. In all experiments the weights are set to zero to avoid altering network dynamics, while the activity of the network is controlled through a single parameter: the current $I_{injected}$, and keeping $I_{E/I}$ at zero. As all the computational

Table 4.1: Neural and synaptic parameters used in the LIF experiments.

Parameters	Values	Units
τ_m	64.0	ms
V_{init}	$[-65.0, -125.0]$	mV
V_{reset}	$[-90.0, -125.0]$	mV
V_{thres}	$[-50.0, -60.0]$	mV
$\tau_{I/E}$	10	ms
$\tau_{refract}$	3	ms

Table 4.2: Izhikevich model parameters. Bracketed parameters indicate the uniformly distributed range with a constant random seed.

Parameters	Values	Units
a	0.02	–
b	0.2	–
c	-65.0	mV
d	8.0	–
V_{init}	$[-65.0, -125.0]$	mV
U_{init}	$[-5.0, 5.0]$	mV
V_{thres}	$[-50.0, -60.0]$	mV
$\tau_{E/I}$	10.0	ms

steps linked to the evaluation of an incoming spike (described by equation 2.12) are the same regardless of the value of the weight itself, this procedure enables full control of the network dynamics through a single parameter. Direct comparison between simulations and extrapolation of the power directly related to neural equation solving and to synaptic events are possible, as presented in Section 4.2.

The first network, illustrated in Figure 4.1, comprises a series of populations each on a single core, recurrently connected in an all-to-all fashion. Upon reaching its threshold, a neuron emits a spike (MC packet) which the router routes back to the originating core, triggering a "packet received" event. Populations are replicated across the 48 chips (764 cores), filling the system, and population activity is controlled by varying $I_{injected}$. This network configuration is used to test local connections within a single SpiNNaker chip with different activity patterns and numbers of neurons and synapses.

The second network introduces inter-chip communications by having each population connected in an all-to-all fashion to n other populations. This is illustrated in Figure

4.2 where each population receives inputs from five other, randomly chosen populations. The network used for this experiment therefore tests inter-chip communication by introducing long-range random connectivity. As in the previous network, the model is extended to run on 768 cores and the network dynamics controlled by varying $I_{injected}$.

4.1.3 Monitoring of the Simulations

During simulation information relative to the status of the experiment is recorded at fixed time intervals; this is needed to verify the correctness of the results and to determine the limiting factors of the system. The recorded data can be downloaded during the simulation or after it has completed. The rest of this subsection describes the methodology used.

Recording the processor utilisation is non-trivial. Each processor cannot monitor its own utilisation, as it would be active when it polled itself, thus appearing 100 percent utilised. A technique was therefore developed that utilises the 2nd timer of SpiNNaker's ARM9 cores, which are otherwise unused. This counter is set up to operate at the processor clock rate (200 MHz) but is disabled at the simulation's start. Whenever an interrupt is received the processor awakens and its first operation within the Interrupt Service Routine (ISR) enables the counter. The counter is updated until the processor is put to sleep, where it is disabled. Therefore the counter accumulates the number of cycles during which a processor has been active. By reading, then resetting, the counter periodically the activity of the local processor may be determined.

The cumulative difference between the total "MC packets" and "DMA Done" event counters per core is also saved at the beginning of a timer event. During that period all interrupts are disabled to ensure that these counters do not change during sampling. This guarantees that all spikes are correctly serviced within the millisecond timer interrupt; occasionally, if a core is busy, a spike might be computed in the next timer interval; simulations where more than 0.1% spikes were not serviced in the correct millisecond were discarded. Results were compared against the NEST simulator (described in section 3.2.1) and the firing rates of the populations were found to be identical. Moreover, in Appendix B there is an execution time comparison between SpiNNaker and multi-threaded NEST for the benchmarking networks presented in this chapter.

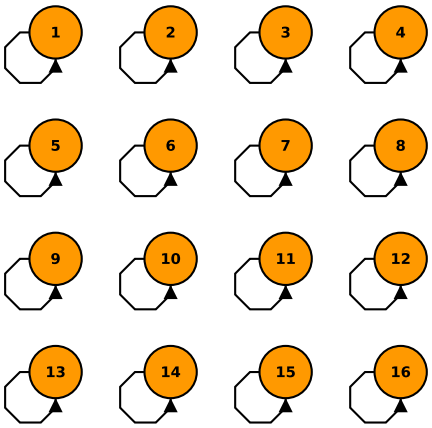


Figure 4.1: Topology for the network used to test local connections.

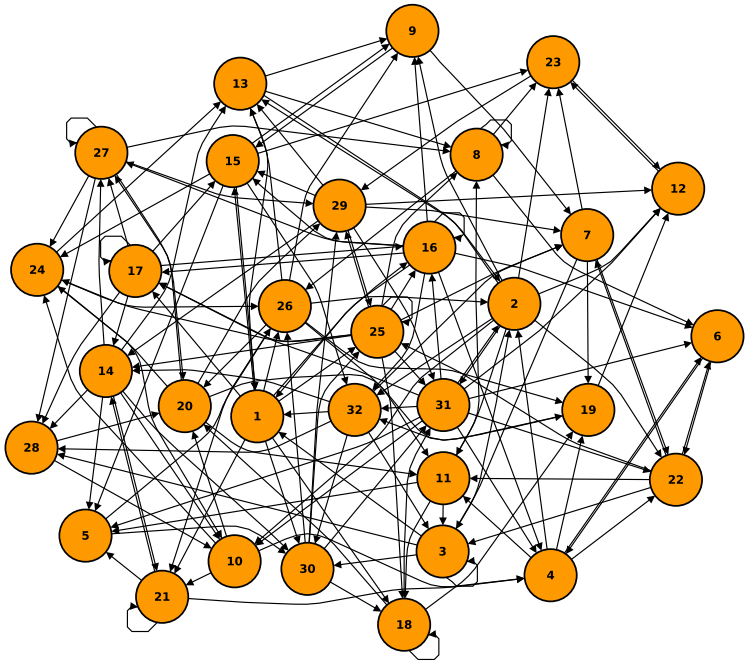


Figure 4.2: Topology for the network used to test random connections.

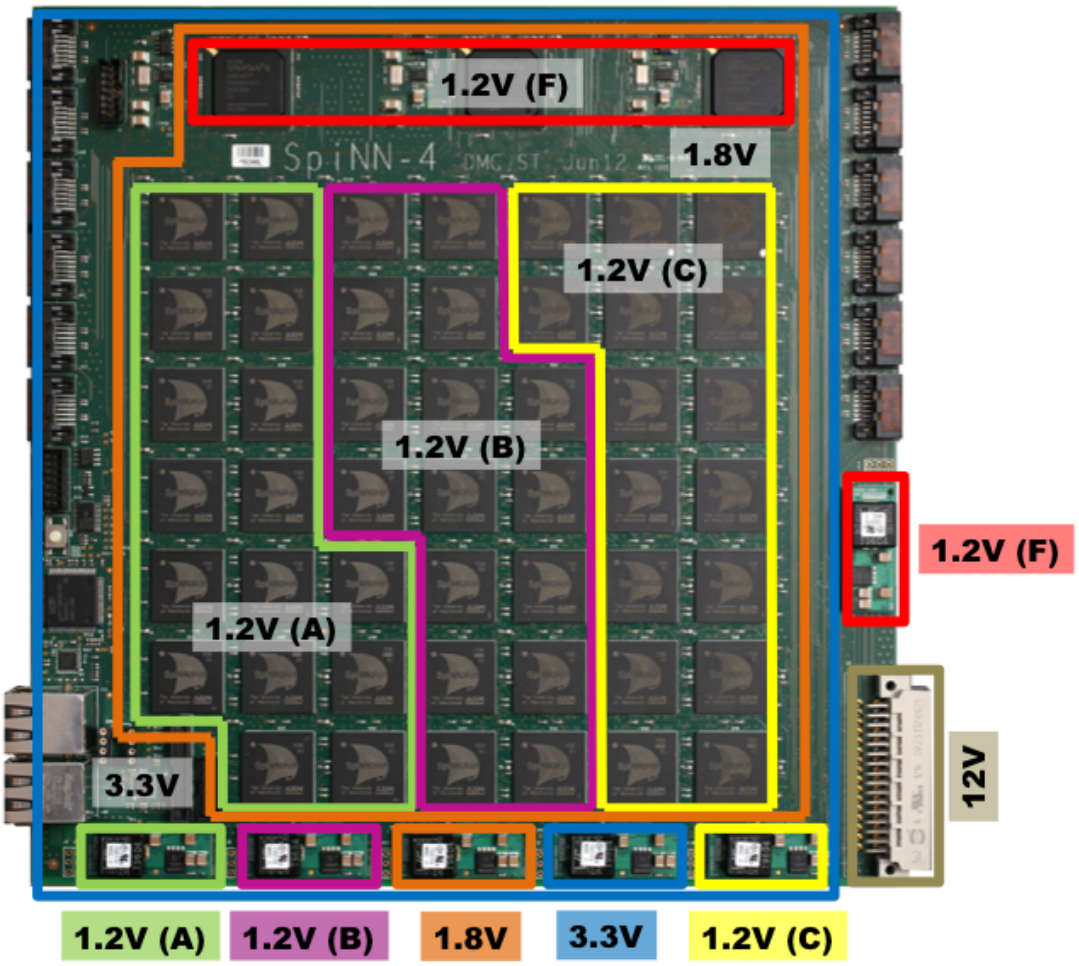


Figure 4.3: Power Distribution of the SpiNN-4 board.

4.1.4 Power Monitoring

Each SpiNN-4 board has 48 SpiNNaker chips plus ancillary components, and is the building block from which larger machines are constructed.

The 48-chip SpiNNaker board has 6 DC/DC converters that supply power to the on-board components. To measure the power dissipation of the board, shunt resistors were placed in series with the 6 DC/DC converters: 0.03 Ω resistors were placed in series with the 1.2 V regulators (A, B, and C in Figure 4.3) that supply the SpiNNaker chips. In addition, 0.1 Ω shunt resistors were placed in series with the 1.8 V regulator that supplies the SDRAMs and the 6 bi-directional links of the chips, the 1.2 V regulator that supplies the 3 FPGA chips, the 3.3 V regulator that supplies voltage to the Board Management Processor (BMP), the Ethernet circuitry, the indicator LEDs and the FPGAs, and finally with the 12 V supply to the SpiNN-4 board.

Tenma 72-7750 and Fluke 77 multimeters were used to measure the voltage across the shunt resistors, yielding the current flow through that regulator. The 12 V measurement point, as well as an overall indication of the power consumed by the board, serves predominantly as a check and balance, where the heat generated by the shunt resistors and the efficiency of the DC/DC converters were taken into account.

An additional measure was employed to assist in the verification process. A Model 2000MU-UK Wattmeter connected in-line with the mains supply was used and before the switched-mode AC/DC adaptor. This meter displays a second by second integrated display of the power passing through it, and therefore gives an overall power including all losses in all transformers, shunts and the SpiNNaker board's consumption. While it was not anticipated that this would be particularly accurate, by using a straight line 80% efficiency factor for the 12 V AC/DC converter in use, the meter tended to be within 1 W or 2 W of the measurements calculated using the more accurate calibrated equipment.

4.2 Power and Latency Profiling

4.2.1 Power Characterisation

This section describes a series of experiments conducted using large-scale simulations of spiking neurons in biological real-time using a SpiNNaker board with 48-chips (SpiNN-4). The SpiNN-4 board contains 864 ARM processors, of which 768 are used for neural applications, 48 for monitoring and 48 as spares for fault-tolerance purposes

[Furber et al., 2013]. The aggregate memory of the board exceeds 6 GBytes, distributed across the chips and cores.

Unless specified elsewhere the following configuration has been used: the "packet received" callback priorities, used for initiating a DMA transfer of the relevant synaptic information from the chip's SDRAM, were set to minus one (pre-emptive), to ensure that packets were cleared from the communications controller immediately upon receipt. The "DMA done" callback priority, which is used for updating the status of each synapse (its weight and delay) was set to zero, which is non-queueable and directly executed by the scheduler. The "timer tick" callback priority, for solving the neural equations and updating the synaptic currents, was set to two (lowest queue-able priority). The processor clocks were set to 200 MHz, routers and system buses to 100 MHz, while the off-chip memory clocks to 133 MHz.

A series of experiments were conducted to characterise the neural and synaptic models, formulating a power estimation model for the SpiNNaker system, based on the number of neurons, synapses and their mean firing rates.

To monitor the power consumption during experiments a Tektronix TDS 3034B oscilloscope and a FLUKE 77 multimeter were used to measure the voltage across shunt resistors in series with the 1.2 V and 1.8 V voltage regulators which supply the SpiNNaker chips and their SDRAMs respectively, see Figure 4.3.

These measurements provided a detailed insight into how much power is consumed by the chips for different states of execution. In a previous study of a biologically plausible model of a cortical column [Sharp et al., 2012], a similar approach was used to measure the energy required per neuron on an earlier generation 4-chip SpiNNaker board. In this section, however, the main focus was to develop a more controlled and systematic simulation environment to obtain a more general SpiNNaker power characterisation.

Areas of particular interest were the power consumed by the chips after reset taken after a power cycle and after loading the API while executing an empty timer callback, without any neural or synaptic computation. This latter power measurement will be referred to as the baseline for the remaining subsections. To measure the energy required per neuron per millisecond of simulation time for the LIF and Izhikevich model, the first benchmark network (Figure 4.1) was used and $I_{injected}$ set to zero. Figure A.2 shows the instantaneous voltage across the shunt resistor placed in series with the 1.2 V (A) voltage regulator, for a variable number of neurons per core, while $I_{injected}$ was set to zero. When calculating the energy per synaptic event, which occurs whenever a spike

arrives at a synapse [Sharp et al., 2012], the synaptic weights were set to zero thus ensuring the dynamics of the network were not altered by outputting spikes. These set of measurements allowed the formulation of a model of power consumption for the SpiNNaker platform and demonstrated how it varies relative to synaptic events and numbers of neurons and synapses.

The fixed and variable power consumption can be characterised in terms of:

- **Idle Power (P_I)**: the power used by a SpiNNaker board after the boot state, with no application running.
- **Baseline Power (P_B)**: the power used when operating SARK and the SpiNNaker API, calculated by loading a neural kernel containing no neurons, where timer events are operating but have no actions.
- **Neural Power (P_N)**: the power required to simulate a neuron (the energy used to solve the equation for a neuron with a millisecond time step), which can be used to estimate the overall power consumption of a model comprising n neurons.
- **Synaptic Power (P_S)**: the power associated with the activation of neural connections (synaptic events), used to estimate the power consumption related to network activity s.

The overall power consumption can therefore be described as:

$$P_{tot} = P_I + P_B + (P_N \cdot n) + (P_S \cdot s) \tag{4.1}$$

To estimate the power for the LIF and Izhikevich neurons, the locally and randomly connected network models were run while disabling spike transmission. As a consequence "packet received" and "DMA done" events were not triggered, the only activity in the network being caused by $I_{injected}$, which controlled the firing rates of the populations. The difference between the baseline power and this simulation is solely attributed to the "timer tick" event solving the neural equations.

To estimate the power related to synaptic events, spike transmission and elaboration is re-enabled, and all weights are set to 0 allowing them to be computed by the callbacks with no impact on the network activity. In this context the number of synaptic events can be measured and the power increment between this simulation and the one used to estimate the neural power determined.

In the results presented in Table 4.3 it is noticeable that for both benchmark networks, the CPU utilization tracks the power consumption.

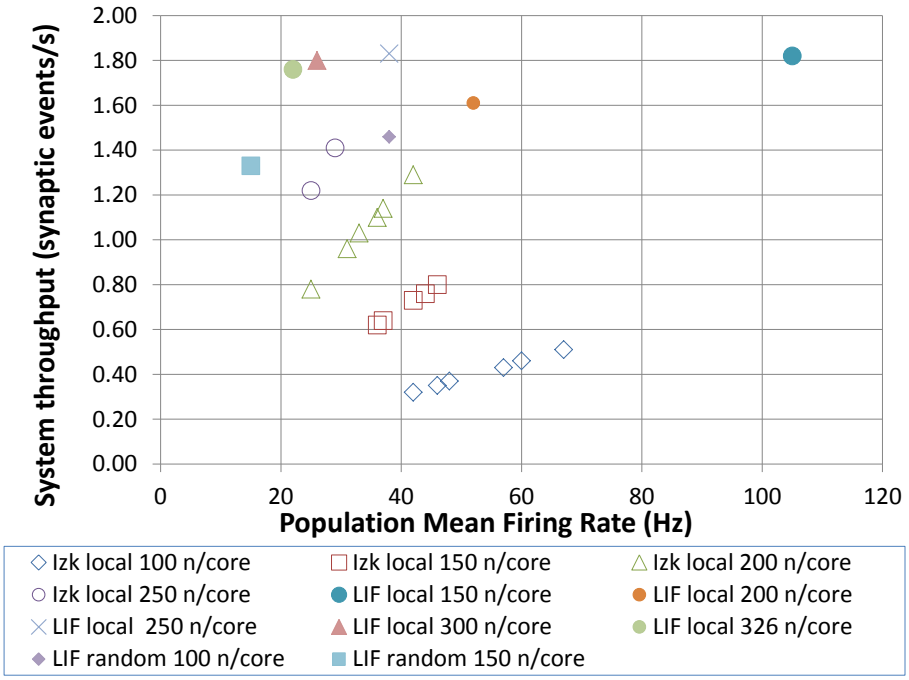


Figure 4.4: Performance for the networks presented.

4.2.1.1 Locally-connected network

In a locally connected network every population, comprised of n neurons, is recurrently connected with $n \times n$ synapses. $I_{injected}$ is used to control the firing rate f of the population; the total number of synaptic events s associated with a population can then be calculated as $s = n \times f \times n$, where every neuron n of the population fires f spikes a second, each activating n connections. The performance estimation assumes that synaptic events dominate the simulation, and are the bounding limit of the platform. Figure 4.4 shows the measured number of synaptic events running on the 48-node board for different-sized populations of Izhikevich and LIF neurons. Results have been divided according to their neural type (Izhikevich or LIF), network topology (local or random) and the number of neurons. For the Izhikevich neural model (outlined markers) a number of different experiments with the same activity, but increased number of neurons, are presented, while for the LIF neuron only the top example for each population size is presented. Results of the top six simulations are reported in Table 4.3, which includes networks up to 200 K Izhikevich neurons and 250 k LIF neurons, with over a billion synaptic events per second simulated using less than 1 Watt per SpiNNaker chip.

Thanks to the regularity and controlled activity of the proposed neural networks, the power characterization for these models is straightforward – added to the baseline

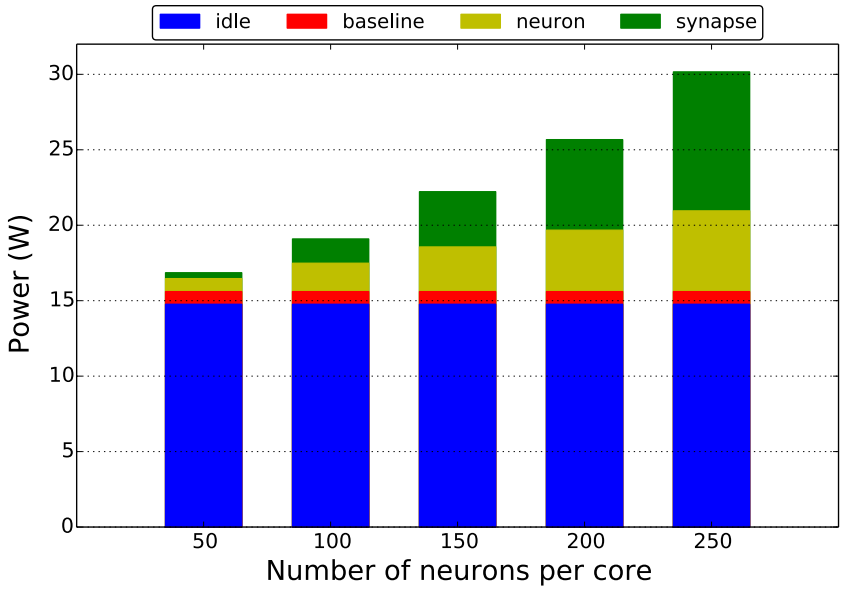


Figure 4.5: Power measurements for the locally-connected network using Izhikevich neurons and a fixed firing rate of 25 Hz.

power costs there is a linear cost associated with the number of neurons simulated, and a quadratic cost of synaptic events which varies with the square of the number of neurons multiplied by the firing rate. This model is illustrated in the power measurements reported for a locally-connected network of Izhikevich neurons, plotted in Figure 4.5, where the firing rate remains constant and the number of neurons was varied. From this figure it can be seen that the power consumption associated with solving neural equations is indeed a linear function (as shown by the third yellow part of the figure), whilst the power associated with the synaptic events (green, fourth part) grows quadratically with the number of neurons.

4.2.1.2 Randomly-connected network

To describe the platform performance with more complex interconnectivity models the second topology described in Section 4.1.2 was used (Figure 4.2), where each population receives all-to-all connections from five randomly chosen populations. With each neuron receiving $5 \times n$ connections, the number of synaptic events $s = 5fn^2$. Here spikes can be routed from any chip in the 48-node board to any other, as connections are randomly picked – creating short, mid-range and long connections. The power related to neurons and synaptic events was calculated as in the previous experiments, finding similar values; these simulation results are also listed in Table 4.3.

Table 4.3: Simulation results for both benchmark networks.

neural model	LIF	LIF	Izk	Izk	LIF	LIF
network topology	local	local	local	local	random	random
neurons per core (population)	250	326	250	250	150	100
synapses per core	62,500	106,276	62,500	62,500	112,500	50,000
synaptic events per core	2,384,500	2,296,996	1,582,500	1,831,969	1,733,250	1,900,000
firing rate (Hz)	38	22	25	29	15	38
total neurons	192,000	250,368	192,000	192,000	115,200	76,800
total synapses (million)	48	81.62	48	48	88.2	39.2
total spikes/s	7.3	5.4	4.9	5.6	1.7	2.9
total synaptic events (billion/s)	1.83	1.76	1.22	1.41	1.33	1.46
overall power (W)	35.39	36.37	30.16	32.08	26.81	26.91
average power/chip (W)	0.74	0.76	0.63	0.67	0.56	0.56
power per neuron (µJ/ms)	26	27	27	28	22	24
energy per synaptic event (nJ)	8	8	8	8	7	6
CPU utilization	62%	66%	49.7%	54%	37%	37%

4.2.1.3 Energy per Multicast Packet

In the SpiNNaker system spikes are transmitted as 40 or 72-bit multicast (MC) packets implementing the Address-Event Representation (AER) [Mahowald, 1994] scheme, where the information transmitted is the address of the firing neuron. Each packet comprises an 8-bit packet header and a 32-bit routing key identifying the neuron that fired; there is also an optional 32-bit payload not normally used for neural applications [Wu and Furber, 2010].

To measure the energy consumed by the router per MC packet and per SpiNNaker link while sending a packet to a neighbouring chip, an additional experiment was conducted using a board with a single SpiNNaker chip, as shown in Figure 4.6. The advantage of using this board is that the 6 bi-directional SpiNNaker links can be wrapped around, as shown in Figure 4.7. A single ARM9 core was used and a single routing entry was set, broadcasting a single MC packet to all 6 bi-directional links simultaneously. Thus the number of packets processed by the router grows exponentially for every new packet the router receives, and the router should be able to achieve its maximum bandwidth without any additional effort from the ARM9 cores. Upon sending the single MC packet the ARM9 core goes back to sleep and wakes up after 60 seconds to terminate the simulation and report the total number of MC packets that have been processed by the router.

A 0.05 Ω shunt resistor was placed in series with the 1.2 V voltage regulator which supplies the SpiNNaker chip and an 1 Ω resistor was placed in series with the 1.8 V voltage regulator which supplies the SDRAM and the inputs/outputs of the chip, as shown in Figure 4.6. A BK Precision 2831E voltage meter was used to measure the voltage across the resistors, sampled once per second.

The experiment and the baseline were run for 5 trials each, while the voltage meter recorded the voltage across each shunt resistor. During the baseline, the ARM9 core does not transmit the single MC packet to the router.

For the baseline the mean voltage across the 0.05 Ω shunt resistor was 11.084 mV, while the standard deviation from the mean 3.53 μV. The mean voltage of the 1.0 Ω shunt resistor was 24.43 mV, while the standard deviation from the mean was 0.14 mV. During the experiments, the mean voltage across the 0.05 Ω resistor was 12.40 mV and the standard deviation from the mean 4.15 μV, while the total number of MC packets the router processed was 1.7 billion packets, Table 4.4. The mean and standard deviation voltage across the 1.0 Ω resistor was 70.0 mV and 0.09 mV, while the total number of MC packets the router issued over the 60 second simulation period, were 274 million

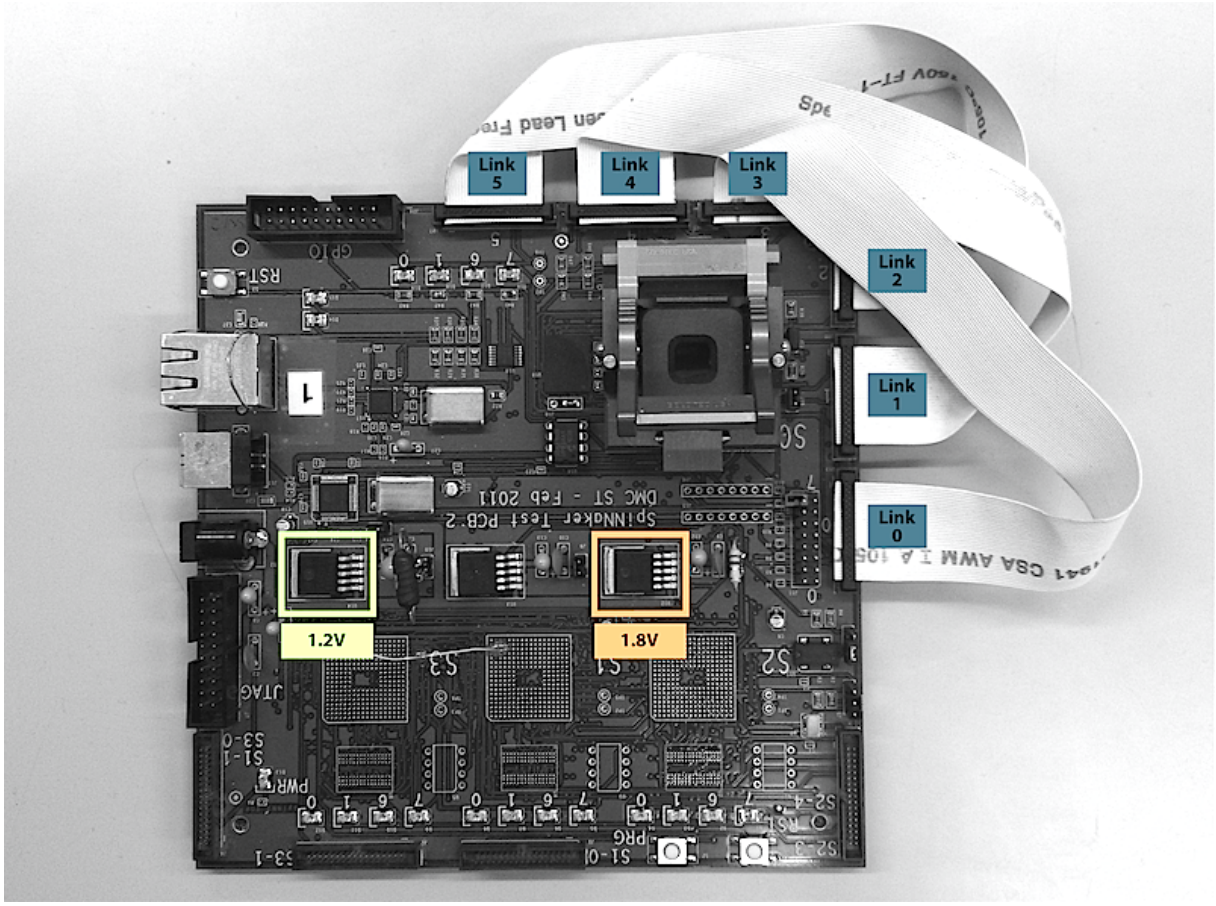


Figure 4.6: Single SpiNNaker chip board.

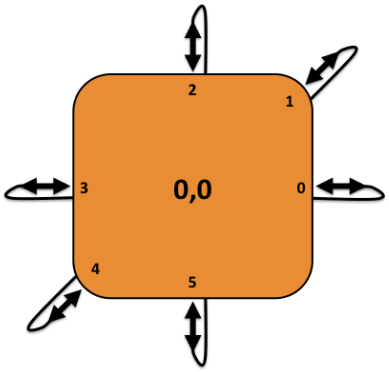


Figure 4.7: Connectivity of the 6 bi-directional links on the single SpiNNaker chip board.

Table 4.4: Power dissipated by the SpiNNaker chip for both the baseline and during the experiment.

	Baseline	**Experiment**	**Units**
Power Dissipation	0.266	0.297	W
MC Packets	0	1,704,110,496.6	-

Table 4.5: Power dissipated by the SDRAM and the inputs/outputs of the SpiNNaker chip for both the baseline and during the experiment.

	Baseline	**Experiment**	**Units**
Power Dissipation	0.044	0.126	W
MC Packets	0	274,674,145.8	-

packets, see Table 4.5.

By utilising Ohm's law the current drawn by the chip during the experiments and the baseline were calculated. The current drawn by the 1.2 V voltage regulator during the baseline was 11.084 mV ÷ 0.05 Ω = 0.22 A, while during the experiment 12.40 mV ÷ 0.05 Ω = 0.24 A. The current drawn by the 1.8 V voltage regulator during the baseline was 24.43 mV ÷ 1.0 Ω = 0.044 A, while during the experiment 70.0 mV ÷ 1.0 Ohm = 0.07 A.

Similarly, the power drawn by the chip during the baseline and the experiment can be calculated. For the 1.2 V voltage regulator the power drawn during the baseline was 0.22 A · 1.2 V = 0.266 W, while during the experiment 0.24 A · 1.2 V = 0.297 W. For the 1.8V voltage regulator the power drawn during the baseline was 0.024 A · 1.8 V = 0.044 W and during the experiment 0.07 A · 1.8 V= 0.126 W. The power dissipation results are summarised in Tables 4.4 & 4.5 for both the baseline and the experiments.

The difference between the power dissipated during the experiment and the baseline is relevant to the power dissipated by the router to process 1,704,110,496.6 MC packets during the 60 seconds of the experiment. This is 0.297 W − 0.266 W = 31.77 mW. Similarly, the power dissipated by the 6 links to transmit and receive 274,674,145.8 MC packets during 60 seconds is 0.126 W − 0.044 W = 81.98 mW.

The energy consumed by the router per MC packet can be calculated as follows:

$$E_{packet} = \frac{31.77 \text{ mW} \cdot 60 \text{ s}}{1,704,110,496.6 \text{ Pkts}} = 1.11 \text{ nJ} \tag{4.2}$$

while the energy consumed by a single SpiNNaker link to transmit and receive a MC packet to is:

$$E_{link} = \frac{81.98 \text{ mW} \cdot 60 \text{ s}}{274,674,145.8 \text{ Pkts}} \cdot \frac{1}{6} = 2.9 \text{ nJ} \tag{4.3}$$

Results are summarised in Table 4.6. A router clocked at 100 MHz consumes 1.11 nJ per MC packet, or 28 pJ per bit, while energy required by an external link to transmit a MC packet is 2.9 nJ.

Table 4.6: Energy consumed by the SpiNNaker router and external links per MC packet.

Parameters	Values (nJ)	Description
E_{packet}	1.11	Energy consumed by the router per MC packet
E_{link}	2.9	Energy consumed by a link to transmit a MC packet

4.2.2 Intra- and Inter-Chip MC Packet Latency

At the heart of the SpiNNaker chip lies a packet-based multicast (MC) router [Wu and Furber, 2010], responsible for communicating spikes to its internal cores or to other chips via six asynchronous bi-directional links. Its pipelined implementation enables it to route one packet per clock cycle to all or a desired number of output links in an uncongested network. If any of the output links are busy, however, the router will try to reroute the packet at every clock cycle until it reaches a predefined number of clock cycles. Failing this it will attempt emergency routing via the link which is rotated one link clockwise from the blocked link (not applicable for destinations internal to a SpiNNaker chip). Similarly, if the emergency route fails, the router will retry emergency routing at every clock cycle until it reaches a second user-defined number of cycles when it finally drops the packet.

This section describes a series of experiments conducted to investigate the intra- and inter-chip MC packet latencies as a function of the router's waiting time and synthetic traffic going through a link. For these experiments, a parametrised software was developed using the SpiNNaker API. The "packet received" callback priorities were set to be pre-emptive to ensure that packets were cleared from the communications controller immediately upon receipt. The "timer tick" callback priority, which was used by the cores for terminating the simulation after 60 seconds, was set to non queueable priority. Finally, the priority of the callback function developed to generate MC packets was set to the lowest queueable priority. The processor clocks were set to 200 MHz, routers and system buses to 100 MHz, while the off-die memory clocks to 133 MHz.

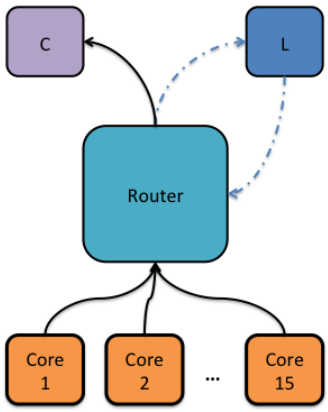


Figure 4.8: Block diagram showing the topology used to measure the intra-chip packet latencies.

4.2.2.1 Intra-chip MC packet latency

The first experiment was aimed at demonstrating the core-to-core packet latency within a SpiNNaker chip as a function of a congested internal link and the router's wait time before dropping a packet. A congested link means that a core has received more packets than it can process on time and a back-pressure signal will propagate to the router through that link. For this experiment 17 cores were used, one dedicated to measuring the core-to-core MC packet latency within a SpiNNaker chip, by sending a packet to the router every 500 ms using the "timer tick" callback function and receiving it back. The second hardware timer, available on each core, was used to count the clock cycles from sending to receiving the MC packet (with nanosecond resolution). Each of the remaining 15 cores would generate approximately 1.7 million packets per second, which would be routed to one particular consumer core (C) whose sole task was to count the received packets, see Figure 4.8. The logged MC packet latencies during the simulation, the values of the software counters and additional diagnostic information from the router were then uploaded to the SDRAM when the simulation was over and fetched by the host for further analysis.

Figure 4.9 shows the mean and standard deviation of the intra-chip MC packet round-trip delay time (RTD) as a function of the total number of MC packets per second the router has issued to the consumer core (C) and for various router wait times. What can be observed from this figure is that, in an uncongested network, the intra-chip round-trip delay time is constant at 0.825 µs, see Figure 4.9(a). Within this time is included the software overhead of the SpiNNaker API required to write the MC packet

to the communication controller, the time needed for the packet to traverse through the internal link to the router, the time required for the packet to go through the router and again through the internal link to the communication controller of the target core and finally the software overhead of receiving the packet from the communications controller. The aforementioned times can be expressed as:

$$t_{\text{RTD}}^{\text{Intra}} = t_{\text{Send}}^{SW} + 2 \cdot t_{\text{Link}}^{\text{local}} + t_R + t_{\text{Receive}}^{SW} \quad , \tag{4.4}$$

where t_{Send}^{SW} is the software overhead of the API required to write the key of a MC packet to the communications controller, t_{Receive}^{SW} is the time passed from handling the interrupt raised by the communications controller to branching to the callback function assigned to handle the packet received events, t_R is the time required by the router to process a single MC packet, and finally $t_{\text{Link}}^{\text{local}}$ is the time a single MC packet needs to go through the local links.

The t_{Send}^{SW} and t_{Receive}^{SW} times were found to be $t_{\text{Send}}^{SW} = 0.415$ µs and the $t_{\text{Receive}}^{SW} = 0.13$ µs, by utilising the second hardware timer. Assuming that the time consumed by a MC packet to traverse through the local links is much smaller than the time spent by the router to process a packet, $t_{\text{Link}}^{\text{local}}$ can be ignored. Solving equation 4.4 for t_R, the time the router requires to process a single MC packet is 0.28 µs.

As soon as congestion occurs, which for this experiment happens when the consumer core (C) receives more than 3.6 million packets per second, the communications controller of the consumer core (C) starts adding back-pressure on the router, which attempts to resend the packet at every clock cycle until it reaches a predefined number of cycles (240 default) at which point the packet is finally dropped, see Figure 4.9(b). This back-pressure signal propagates back along the pipeline and the router stops receiving new packets until back pressure has been released [Wu and Furber, 2010]. As a consequence, the MC packet latency increases and the hardware buffers of the communication controllers of the cores generating the MC packets are not emptied; this explains why the total number of generated packets plateaus, as seen in Figure 4.9(c), while failed packets increase (software buffer full), see Figure 4.9(d). When the router's waiting time is set to 240 cycles (default) and 60 cycles no packets were dropped in any of the trials but the worst-case round-trip delay time went up to 6.5 µs. For router wait times of 20 cycles, and below, the worst-case round-trip delay time drops below 4 µs but the total number of dropped packets per second increases dramatically. This

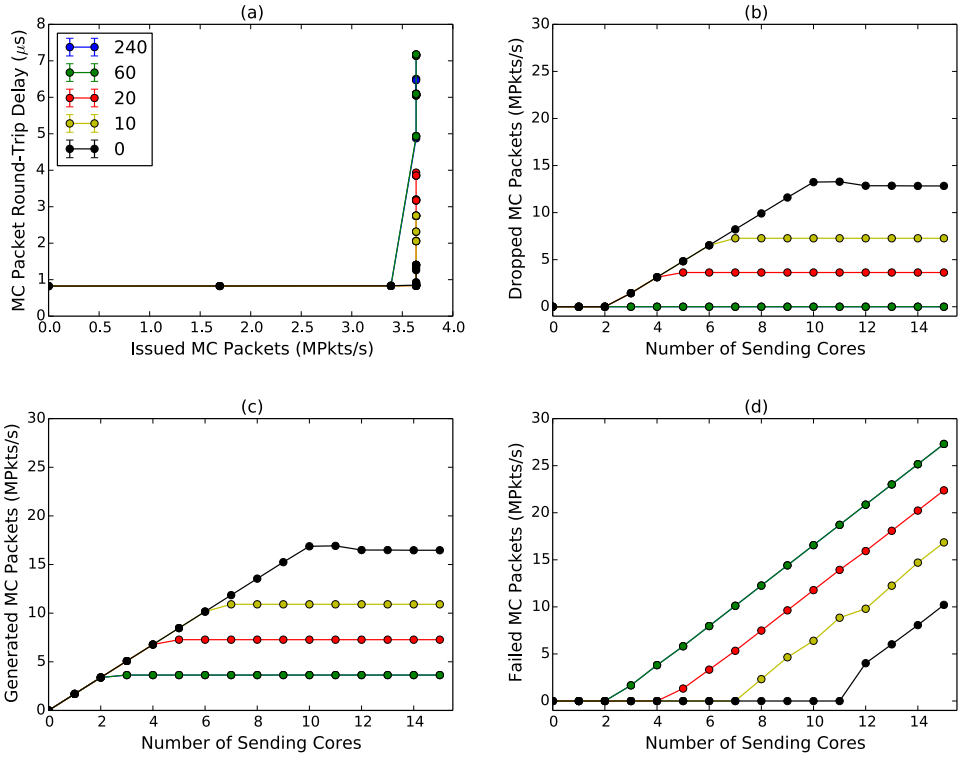


Figure 4.9: Intra-chip MC packet RTD times as a function of the total number of packets and for different router wait times.

trade-off between intra-chip MC packet latency, packets being dropped or not being sent to the router at all, requires further investigation as it depends on the requirements of a particular application.

A core clocked at 200 MHz may execute 200 million single-cycle instructions per second. When the core receives a packet it is processed by an interrupt service routine (0.13 µs); this copies the MC packet from the communication controller and then branches to the particular callback function assigned to handle a packet received event, which for this experiment is a 5 assembly instruction routine which increments a software counter. Based on this information, in theory, the consumer core (C) should be able to receive approximately 6.4 million packets per second. However, as seen in Figure 4.9, an ARM9 core cannot receive more than 3.6 million packets per second. The maximum throughput of a SpiNNaker router, clocked at 100 MHz, is 100 million MC packets per second [Wu and Furber, 2010] so it is clearly not the bottleneck in this experiment.

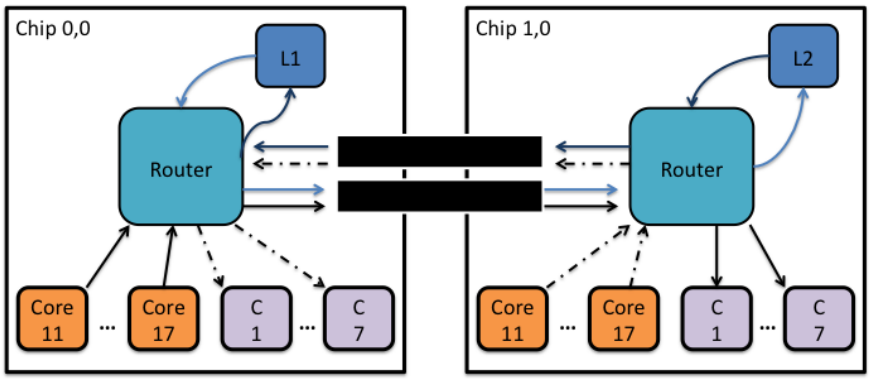


Figure 4.10: Block diagram showing the topology used to measure the inter-chip MC packet latency as a function of synthetic traffic going through a link.

4.2.2.2 Inter-chip MC packet latency

The router of a SpiNNaker chip can communicate packets to neighbouring chips through six self-timed bi-directional links. The average bandwidth (transmit/receive) of each link is 6 million packets per second (240 gigabits per second) and this may vary with the temperature, voltage or silicon properties. An experiment was conducted to determine the inter-chip RTD of a MC packet transmitted through one of the 6 bi-directional links as a function of the link's outgoing and incoming traffic. For this experiment a core (L1) generates a MC packet every 500 ms and the router routes it to a neighbouring chip through one of the six bi-directional self-timed links. Upon receiving the packet the second router would route it to a particular core (L2), whose sole task was to change the key of the packet and retransmit it back to the router which had an appropriate routing entry to route it back to the originating core, Figure 4.10.

Seven cores on each chip were used to generate packets which were routed to seven consumer cores (C) on the adjacent chip following a one-to-one mapping. This way the total number of packets per second a consumer core receives remains below 3.6 million packets per second, (which is the maximum number of packets a core can receive) ensuring that no additional pressure is added to the routers.

The generated packets were controlled by an inter-packet interval (IPI) parameter, which is a delay in microseconds before transmitting the next MC packet. Results are presented as the percentage of the utilisation of the incoming and outgoing packets going through a link per second, with 100% utilisation meaning 6 million packets per second.

The mean and standard deviation of the round-trip delay times of MC packets are presented in Figure 4.11. When there is no traffic the round-trip delay time of a MC packet is 2.535 µs. Within this time is embedded the time required for the packet to go through each router twice, twice through the external link, and also two software processing overheads of sending and receiving the packet back to the router. This can be expressed as:

$$\begin{aligned} t_{\text{RTD}}^{\text{Inter}} &= 2 \cdot t_{\text{Send}}^{SW} + 4 \cdot t_{\text{Link}}^{\text{local}} + 4 \cdot t_R \\ &\quad + 2 \cdot t_{\text{Receive}}^{SW} + 2 \cdot t_{\text{Link}}^{\text{external}} \end{aligned} \tag{4.5}$$

where $t_{\text{Link}}^{\text{external}}$ is the time a MC packet requires to traverse through an external link to a neighbouring chip. Solving for an RTD time of 2.535 µs and by using the results of equation 4.4 for t_R and by ignoring the time of $t_{\text{Link}}^{\text{local}}$, the time a MC packet needs to go through an external bi-directional link is 0.1625 µs.

For a link utilisation of 60%, for both outgoing and incoming traffic, there is a 3% increase in the RTD time. When both the incoming and outgoing link utilisation reaches 80% a very small number of packets were dropped as both routers attempted to reroute the packets for the default wait times (240 cycles), hence the dramatic increase in the RTD times. Results are summarised in Table 4.7.

Table 4.7: Experimental results of the SpiNNaker latencies.

Parameters	**Values** (µs)
t_{Send}^{SW}	0.415
t_{Receive}^{SW}	0.13
t_R	0.28
$t_{\text{Link}}^{\text{external}}$	0.1625

In order to investigate the variability of the self-timed bi-directional links of the SpiNNaker platform, additional experiments were carried out on SpiNNaker boards with 48-chips, such as the SpiNN-4, as shown in Figure 4.3, and a SpiNN-5 board,

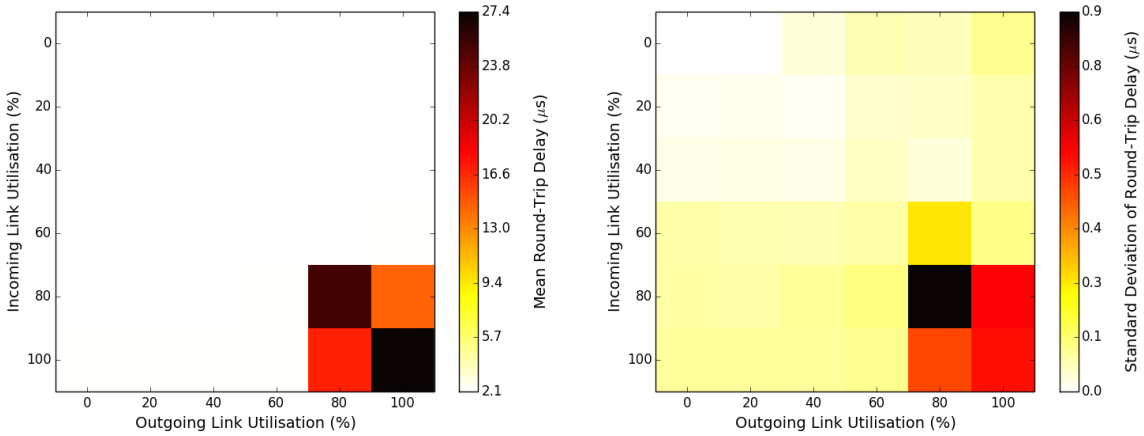


Figure 4.11: Mean and standard deviation of round-trip delay times of MC packets as a function of the percentage of a link's utilisation.

as seen in Figure 3.5. For these experiments a similar methodology was used as in Section 4.2.2.2. This time however the L2 core would reside not on a neighbouring SpiNNaker chip but on a chip across the diagonal of the board thus requiring the MC packets to pass through multiple routers (hops) in order to reach the destination and return to L1.

Results are summarised in Table 4.8 and Figure 4.12. As can be seen from Table 4.8, $t_{\text{Link}}^{\text{external}}$ does not have fixed value but its performance depends on factors such as temperature, voltage or silicon properties due to its asynchronous nature. Finally, Figure 4.12 presents the experimental results retrieved from the SpiNN-4 and SpiNN-5 boards and theoretical results by utilising equation 4.5 and Table 4.7.

Table 4.8: Mean and standard deviation of the external link latency.

	Mean $t_{\text{Link}}^{\text{external}}$	**Std**	**Units**
SpiNN-4	0.0961	0.0266	µs
SpiNN-5	0.0861	0.0208	µs

4.2.3 Discussion

Large-scale modelling of neural tissue by computer simulation is an essential step in understanding how the brain works as demonstrated by the high-profile interest shown by IBM [Arthur et al., 2012] and funding bodies such as the Human Brain Project (HBP) [HBP, 2013]. In the previous sections, SpiNNaker, a project which plays a part within

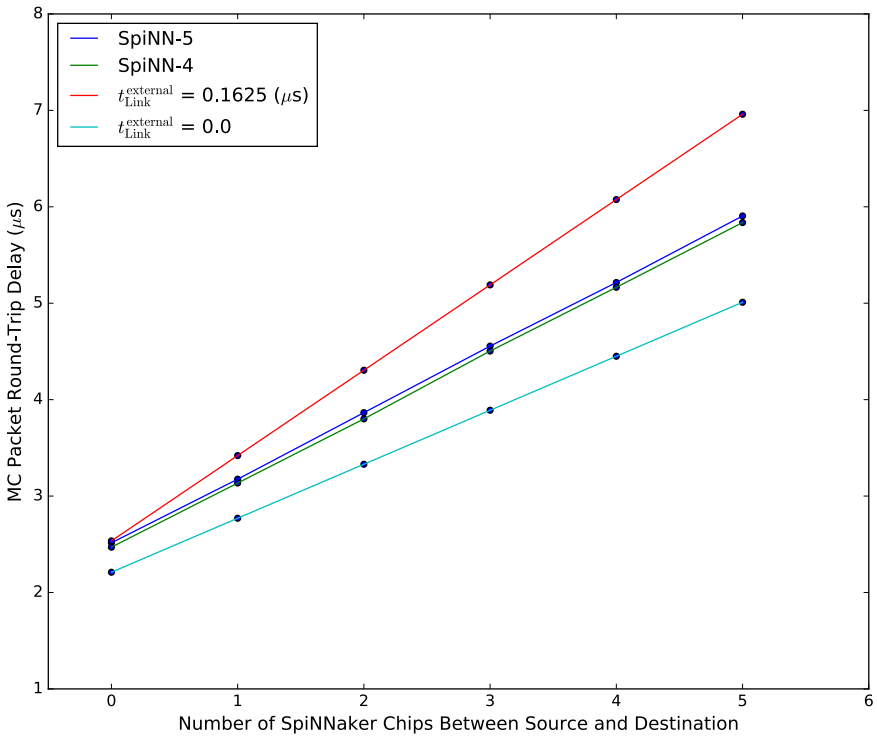


Figure 4.12: Inter-chip MC packet latency as a function of the number of hops. Experiments performed on a SpiNN-4 and SpiNN-5 board.

the HBP, was characterised by analysing its reconfigurability, scalability and power consumption. The SpiNN-4 board used for this study constitutes the building block of much larger SpiNNaker machines. Benchmarking networks have been presented, both with local and long-range connectivity, using the Izhikevich and LIF neural models, and demonstrated the flexibility of the system in terms of neural models, topologies and the dynamical range of activities simulated. Both neuron types and network models were characterised in terms of power consumption, by producing a model describing fixed and variable power costs, relating the latter to the number of neurons modelled in the system and the number of synaptic connections activated each second. More specifically, it was shown that the energy per synaptic event for a current-based instant-rise single-exponential decay synapse is 8 nJ, while the energy required to simulate a neuron with a millisecond time-step is 28 µJ/ms. The energy required by the router to route a MC packet is 1.11 nJ, while the energy consumed by an external link while transmitting a single MC packet was 2.9 nJ. The results show that networks of a quarter

of a million neurons, tens of millions of synapses and dynamic activity of over a billion synaptic events per second can be delivered within a 30 W power envelope (less than 1 W per SpiNNaker chip). Based on Table 4.3, for the random network of Figure 4.2, SpiNN-4 delivers 54.27 million synaptic operations per second (SOPS) per Watt in real-time (millisecond time steps); for the local network (Figure 4.1) SpiNN-4 delivers 51.71 million SOPS per Watt.

An additional investigation of the intra- and inter-chip MC packet latencies revealed that in an uncongested network the intra-chip core-to-core MC packet latency is 0.28 μs, while the inter-chip core-to-core MC packet latency is 0.72 μs which includes the time a MC packet to pass through an external link and two routers, and excluding the software overheads of transmitting and receiving a packet. For the spiking neural network simulations discussed in Section 4.2.1 a core receives, in worst-case, 17 kilopackets per second (Table 4.3), which is well below the maximum number of packets a core can receive (3.6 million packets per second), guaranteeing that no back-pressure signals are propagated to the routers from the cores.

4.3 Optimising the Overall Power Usage

As the size of SpiNNaker machines grows substantially, its energy use begins to scale in proportion. While large neural network simulations become possible, what was once a trivial problem, the heat generated and power used by standalone small boards becomes a significant issue in terms of financial cost and environmental management.

It was also noted that during experiments which investigated the energy consumption of SpiNNaker chips at different stages of neural network simulation [Stromatias et al., 2013], Section 4.2.1, a significant portion of the total power was dissipated during the idle state. The idle state is defined as the power used by the system after booting and while in a state where it is ready to accept a workload.

In this section a systematic investigation identifies which components within a SpiNNaker chip are the most energy-consuming, thus proposing new suspend states which minimise the total energy use. This will have a significant impact on the overall energy consumption of larger SpiNNaker machines, especially for the nodes not participating in a simulation. Additional savings are expected during the uploading and downloading phases of a simulation cycle and further benefits include increasing the life-span of the chips and environmental savings by reducing the energy of the cooling sub-systems.

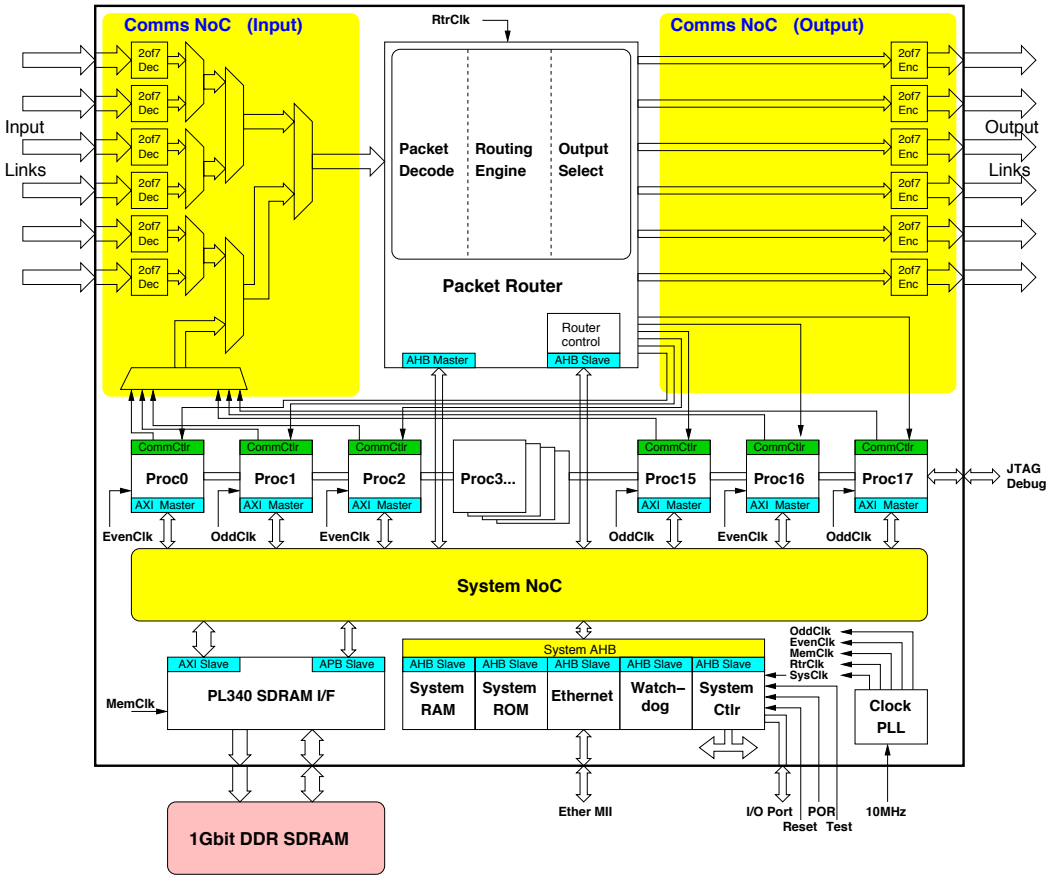


Figure 4.13: Block diagram of a SpiNNaker chip.

4.3.1 Power Profiling the SpiNNaker Chip

There are 5 components within a SpiNNaker chip which require a clock source, the ARM968 cores divided into two banks based on their physical ID, the router, the System AHB bus and the shared SDRAM memory. Each chip receives a 10 MHz input clock from the Board Management Processor (BMP); this can be used as received, further divided by 4 or used as an input to the 2 Phase-Locked Loop circuits (PLLs), as seen in Figure 4.13. Finally, there is an additional clock divider which can optionally divide the input signal, for each of the 5 clock domains, by 2, 3 or 4. The user can control these parameters through the System Controller registers.

A parametrised software was developed using the SpiNNaker API software and SARK (v 1.09). This software made the necessary changes to the SpiNNaker hardware directly, such as peripherals and clocks, resulting in a steady-state environment where accurate and systematic calculations of energy consumption can be made. After the

experiment, the configuration reverts to the standard operating parameters for an idle SpiNNaker system, permitting a direct comparison to be made between the new suspend mode under test and the existing software.

While there are chip-level components which may be individually enabled and disabled including some of the controllers, the router, the PLLs and the individual processor blocks, the dynamic power used by the clocked components was expected to have the largest energy costs. Frequency scaling adjustments should have a big impact on energy use when compared to other components, particularly those that are asynchronous.

At present the run-time software kernel derives all its clock domains from PLL1 set at 400 MHz with the exception of the memory, which is driven by PLL2 at 266 MHz. Processor domains A and B each divide the incoming 400 MHz by two for a 200 MHz clock, and both the router and System AHB bus divide it by four to supply 100 MHz. It is expected that the largest savings can be made by scaling these clocks dynamically while in idle mode even to the extent of shutting down a particular PLL to remove the clock from targeted domains. Extreme measures such as these, however, may have an adverse impact on the recoverability of the component – for example a chip cannot be remotely instructed to exit suspend mode if all its processors are de-clocked and cannot respond.

As indicated above, these experiments will concentrate on the dynamic power of the system and in reducing this to a minimum in both recoverable and non-recoverable states. Where it has been possible to characterise particular peripherals and components, this will be reported in the results section, so that the maximum potential of the proposed suspend modes can be ascertained.

4.3.2 Measuring the Default Idle State of SpiNNaker

The first step towards optimising the total energy consumption is to measure the default idle state, which will serve as a baseline for the remaining experiments. The experimentation is carried out on a single SpiNN-4 board (Figure 4.3) by systematically adjusting a single parameter at a time to ensure that the characteristic information on that component can be gathered. Where it is not possible to alter a single variable at a time, such as where there are combinatorial limitations in PLL assignment, a control experiment is undertaken so that the desired power information can be deduced and recorded.

Using the current SARK software (v 1.09) in the idle state, the following table

records the power at each of the DC/DC converters, which supply power to the SpiNN-4 board. The results are reported in Table 4.9.

Table 4.9: Power dissipation in the default idle state.

DC/DC Converter	Measured Power (W)
1v2 Bank A	5.23
1v2 Bank B	5.17
1v2 Bank C	5.52
1v8 SDRAM	0.90
3v3 Supply	4.67
1v2 FPGA	0.50
Total	**21.99**

While these numbers do not take into account the losses in the converters, they do indicate that the default idle power budget is around 22 W after the conversions to the various required supply voltages. These numbers are used in the experimentation to evaluate the effectiveness of a proposed optimisation.

Since the largest SpiNNaker machine will utilise 1,200 48-chip SpiNNaker boards, it is expected that the power dissipation in the idle state will be approximately 26.4 kW excluding the cooling systems.

4.3.3 Power Dissipation of Clocked Components

The first set of experiments is on the five clock domains of all SpiNNaker chips on a board, and the experiments are devised so that PLL1 is used for controlling the component under test and the remainder of the domains, which are not on test, are clocked from the alternative PLL2.

These clocking domains are as follows: Processor Block A, Processor Block B, The Router, The System AHB (Bus) and Memory.

In the experiments the same configuration is replicated across all 48 chips of a single board machine. As a control methodology, PLL2 is set to 260 MHz and divided by two for the router, AHB and processors, with the memory controller receiving the 260 MHz directly. PLL1 is used for the component under test and is set explicitly to 200 MHz for all experiments with the appropriate divider to meet the target frequency, with the exception of the memory experiments where this is exceeded. Where the target clock is 10 MHz or less, it is sourced from the 10 or 10÷4 MHz clocks directly and PLL1,

although unused, remains switched on and at 200 MHz. The results are summarised in Table 4.10.

If one considers that the total power recorded for the board in the default state (Table 4.9) is 22 W, it is obvious from Tables 4.9 and 4.10 that the majority of the idle dynamic power in the SpiNNaker system is taken by the processor blocks, approximately 70%. The routers and the system busses which also use the 1.2 V supply account for 10% or so, the memory around 8%, and the remainder by other supporting hardware on the board. Clearly it should be possible to attain the largest gains through manipulation of the processor clocks, but it may be possible to make smaller incremental gains by adjusting chip and processor block components when a chip is idle. The 3.3 V supply accounts for the majority of the remaining consumption and thus should also be explored for energy saving opportunities.

4.3.4 Proposed States of Operation

This section suggests a number of new SpiNNaker Suspend Modes (SSMs), both recoverable and non-recoverable, with results shown in Table 4.11. The following subsections provide a brief explanation of each chip suspend mode and its effects on the routing system of a larger SpiNNaker machine.

- **SSM0 – Operational:** This is out of scope for idle power saving, but there is potential in the future to explore this state and seek out frequency scaling strategies to minimise wear out, optimise energy use etc.

- **SSM1 – Wait for Interrupt:** This is the default mode of operation for a SpiNNaker core when it is not in state SSM0. Its context is saved and recovery is on a per cycle basis. This is the current idle mode pre- and post-simulation and does not attempt any further energy management. Cores are operated at 200 MHz, router and system AHB are at 100 MHz and the memory controller 260 MHz.

- **SSM2 – Suspend with SDRAM:** This mode clocks down all processors to the minimum possible frequency 625 kHz (10 MHz $\div$ 4 $\div$ 4). The router is clocked to 50 MHz (PLL1 $\div$ 4) as this provides sufficient bandwidth to cope with a full complement of through external traffic. The System AHB bus is also clocked at 50 MHz in the same way as this provides reliable communications for remote mode change commands. This mode maintains the full memory refresh rate of

Table 4.10: Investigating the power consumption of clocked components. Default values are in bold font.

Component	**330 MHz**	**260 MHz**	**200 MHz**	**100 MHz**	**50 MHz**	**10 MHz**	**2.5 MHz**	**625 kHz**
768 Cores			**15.27 W**	8.09 W	4.43 W	1.14 W	0.38 W	0.20 W
48 Routers			2.57 W	**1.34 W**	0.75 W	0.23 W	0.19 W	0.14 W
48 System AHBs			1.02 W	**0.86 W**	0.75 W	0.59 W	0.43 W	0.36 W
48 SDRAMs	2.36 W	**1.81 W**	1.56 W	0.71 W	0.50 W	0.19 W	0.17 W	0.14 W

Table 4.11: SpiNNaker suspend modes under investigation.

Mode	**Description**	**DC/DC Power Out**	**DC/DC Loss**	**Board Power**	**Power Saving**	**% Saved**
SSM0	Active Operation					
SSM1	Wait for Interrupt	21.99 W	4.85 W	26.84 W		
SSM2	Suspend With SDRAM	7.97 W	2.72 W	10.68 W	16.16 W	60
SSM3	Suspend Without SDRAM	6.40 W	2.00 W	8.40 W	18.44 W	69
SSM4	Node Routing Pass-Through	6.23 W	1.91 W	8.14 W	18.71 W	70
SSM5	FPGA Bypass	5.48 W	1.63 W	7.11 W	19.74 W	74
SSM6	Board Power Down	0.77 W	0.29 W	1.06 W	25.78 W	96

PLL2 at 260 MHz so that when the processors are restored, full context remains available.

- **SSM3 – Suspend without SDRAM:** This mode is identical to SSM2 but stops PLL2, which is fed to the memory controller for refresh. This mode loses external memory context and requires a reconfiguration of the memory controller on recovery.
- **SSM4 – Node Routing Pass-Through:** This mode removes the clock from all clocking domains on a chip (set to a stopped PLL2), except the router which is clocked at 50 MHz (PLL1 $\div$ 4). This way a SpiNNaker board does not become a black hole in the routing fabric since the routers remain in use and full connectivity remains. All context is lost, and the board requires a reset, which may take seconds to initiate and complete, and remote intervention to reboot.
- **SSM5 – FPGA Bypass:** This mode removes all clocks from the SpiNNaker chip clock domains including the router (PLL1 and 2 are disabled). The routing logic must now be handled by the FPGAs which sit on the edges of all the boards.
- **SSM6 – Board Power Down:** This mode removes the power from the board with the exception of the BMP to allow recovery. Both router pass-though and FPGA bypass are not possible and the board is a traffic black-hole.

4.3.5 Power Dissipation During a Full Simulation Cycle

To demonstrate the effects of an optimised suspend mode on the simulation cycle, the SSM2 mode was implemented by adding extra functionality to SARK.

Each time a neural application core participates in a simulation, it sets a bit in a specific place in shared memory, and resets it as soon as the simulation is over. The monitor core polls that memory location periodically to check the status of the application cores and if all application core bits are reset, it enters SSM2. A chip returns back to SSM1 whenever it receives a message from the host, or another SpiNNaker chip, indicating either the uploading of data to the shared memory, downloading of results or the beginning of a simulation.

A simulation comprising 192,000 neurons with 48,000,000 current-based exponential synapses ran in real-time for 30 seconds generating 720,000,000 synaptic events per second. For this simulation the PACMAN tool produced 208 MBytes of synaptic information, neural parameters, neural/synapse models and routing tables, while the size

Table 4.12: The power dissipation of a full simulation cycle for the proposed (SSM2) and default idle mode (SSM1).

Mode	Average Power (W)
SSM1	28.14
SSM2	13.53

of the recorded membrane traces that which had to be downloaded upon the completion of the simulation was 1.2 GBytes.

Current flow was recorded at a rate of 1 Hz from the 12 V supply (Figure 4.3). The same experiment was carried out four times, twice for the new optimised idle mode and twice for the default SSM1 mode to validate the data.

Figure 4.14 shows the power dissipated during a simulation cycle for the default idle state and one with the proposed optimisations. A full simulation cycle on SpiNNaker consists of three phases; uploading the simulation data, running the simulation and retrieving the results. As can be seen in Figure 4.14, all SpiNNaker chips start in SSM2 and gradually the chips which receive simulation data enter into SSM1, returning back to SSM2 as soon as the uploading has finished. The next phase is the execution state, SSM0, where the simulation runs for 30 seconds. When the simulation has ended the SpiNNaker chips enter SSM1, the standard idle mode, and the next time the monitor core polls their status it sets them to SSM2. The last phase is the downloading of the membrane potentials; during this phase only the chip that sends data is in SSM1 and returns back to SSM2 when all data to retrieve is sent back to the host.

The power dissipation for each simulation phase, including the idle state, is presented in Figure 4.15. The simulation which incorporates the new suspend mode, SSM2, is 60% more energy efficient in the idle state, 50% more efficient during the uploading phase and 52% during the downloading phase. Table 4.12 summarises the power dissipation for both simulations. Results indicate that the simulation with the optimised idle mode (SSM2) is overall 52% more energy efficient than the current default one.

4.3.6 Discussion

In this subsection, different approaches towards optimising the overall power dissipation of a 48-chip SpiNNaker board, the building block for creating larger SpiNNaker machines, were investigated. The main focus was on recoverable idle states through dynamic frequency scaling, by systematically examining the power dissipation of each

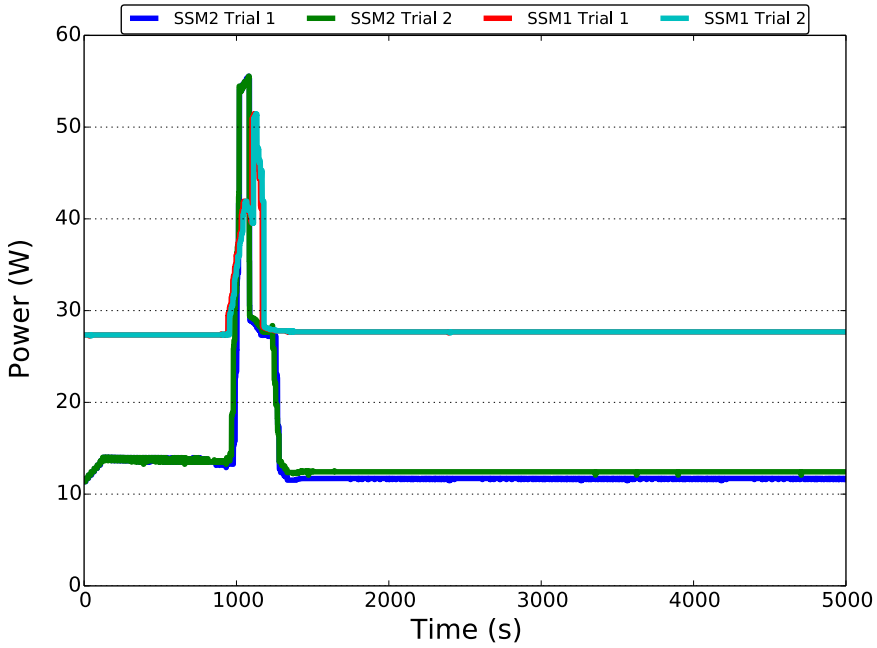


Figure 4.14: A full simulation cycle on a 48-SpiNNaker board using the proposed optimised idle state (SSM2) and the default implementation (SSM1)

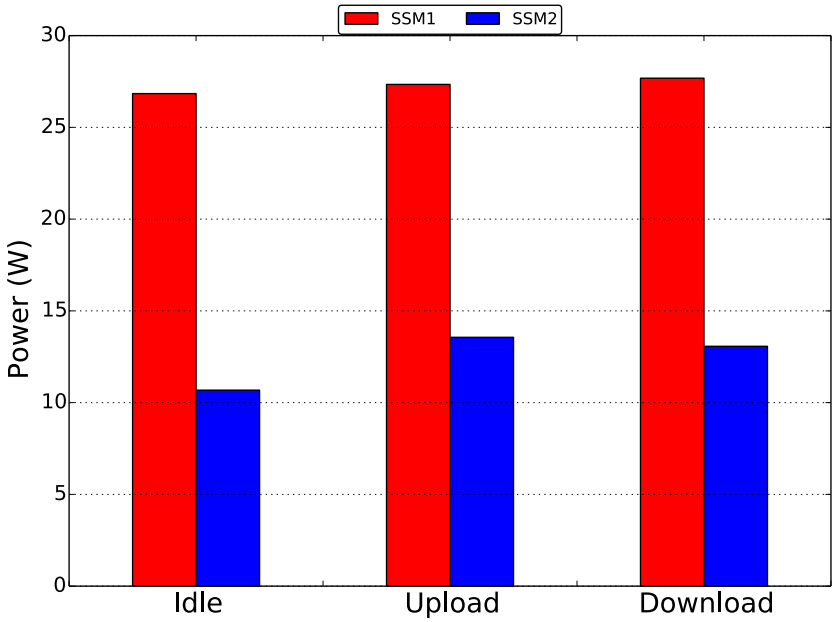


Figure 4.15: Power comparison between the proposed optimised idle state (SSM2) and the default one (SSM1).

clocked component within a SpiNNaker chip. The proposed optimisation was implemented and tested on a large simulation comprising thousands of neurons with tens of millions of synapses with an activity of hundreds of millions of synaptic events per second. The main motivation was to see the effect of the new idle state under a full simulation cycle, which consists of three phases: uploading models and synaptic information, running the simulation and retrieving the results.

The results show that for a SpiNN-4 board the proposed optimisation is 60% more energy efficient for the idle state, 50% for the uploading and 52% for the downloading phase. Moreover, for the same simulation, the power dissipation is reduced by 52%. Based on these findings it is estimated that the final SpiNNaker machine will dissipate approximately 13.2 kW compared to 26.4 kW in the proposed idle state. This will also be beneficial for mobile robotic platforms which utilise SpiNN-4 boards [Galluppi et al., 2014a; Conradt et al., 2014] and have limited power resources.

4.4 Summary

Simulation of large scale neural networks imposes challenges in terms of flexibility, computational performance, communication infrastructure and power consumption. Supercomputers offer great flexibility in that they are fully programmable. Communication between different nodes on such a parallel system can be implemented using the MPI interface [Plesser et al., 2007], but its communication overheads are not ideal for scalable spiking neural network simulations.

Wong et al. [2013] simulated 53×10^{10} neurons with 1.37×10^{14} synapses on the Sequoia - BlueGene/Q supercomputer using Compass [Preissl et al., 2012], a highly-optimised simulator, as part of the DARPA SyNAPSE program. The simulation ran 1,542 times slower than biological real-time and the biggest cost reported was communicating the spikes via MPI messaging. Power dissipation was omitted, but the TOP500 [top] supercomputer list reports that the power used by the Sequoia - BlueGene/Q is 7,890 kW.

Within the DARPA SyNAPSE project IBM recently demonstrated the TrueNorth chip [Merolla et al., 2014b], a digital architecture capable of simulating 1 million neurons with 256 million synapses in biological real-time, while dissipating 63 mW. TrueNorth comprises 4,096 neurosynaptic cores [Arthur et al., 2012], where each core can model 256 single-compartment LIF neurons with 256 axons and 256$\times$256 binary synapses. Each axon can be assigned to four different types, while the synaptic weight

from a presynaptic to a post synaptic neuron is the product of the binary weight, the axon type and a signed integer value stored at the post synaptic neuron. By utilising a similar benchmark neural topology to Section 4.1.2, they reported that TrueNorth is 769× more energy efficient per synaptic event (26 pJ) than SpiNNaker, while the measured computational power in synaptic operations per second (SOPS) was 46 billion SOPS per Watt.

While TrueNorth is intended for real-world applications, SpiNNaker has been designed as an exploration platform for the very active neuroscience research area and for event-based applications interfacing neuromorphic sensors [Galluppi et al., 2012c, 2014a]. Moreover, SpiNNaker is a fully programmable platform, meaning that neurons, synapses and plasticity rules [Jin et al., 2009; Diehl and Cook, 2014; Galluppi et al., 2014b] are implemented in software, whereas in TrueNorth neurons and synapses are implemented in hardware, with the latter being delta-current synapses.

Neuromorphic systems, exploiting sub-threshold transistor dynamics to model neurons in silicon, have been proposed as power efficient modelling systems. These can be scaled to large network models, for example Neurogrid [Silver et al., 2007], a 4×4 system where each *neurocore* node models 65,536 two-compartment cells, tiled in a 256 × 256 array up to a system with a million neurons. Many neuromorphic systems are highly optimized to a particular neural model and offer minimal configurable interconnectivity, often limited by wiring density. Some systems use alternative communication approaches including using an AER packet based infrastructure to enable connectivity and propagate spikes. The HiAER-IFAT framework has been characterized in terms of power for neural and synaptic events [Yu et al., 2012], with multiple chips each modelling 65k bi-compartmental neurons capable of supporting 5 Mevents/s at 50 pW/spike.

The neuromorphic approach can be very power efficient, as neuron dynamics are implemented directly in silicon, but it imposes trade-offs in terms of reconfigurability and scalability. To mitigate such connectivity limits the Brainscales project, which runs networks of millions of synapses in accelerated time, takes the approach of implementing a bespoke packet switched network for its communication requirements.

To overcome the expense and effort of producing a custom chip, some research groups have focused their research on more off-the-shelf, reconfigurable systems. Cassidy et. al [Cassidy et al., 2011] have introduced an FPGA system capable of simulating one million neurons in real time; the system has configurable interconnectivity and uses two 36 Mb SRAM chips, but this ultimately limits the total number of synapses per

neuron. A scalable, configurable real-time system has been recently proposed named Bluehive [Moore et al., 2012], which employs a number of FPGAs interconnected by a packet-switched network. Each FPGA can simulate up to 64k fixed-point Izhikevich neurons with 64 million static delta-current synapses, producing 1 billion synaptic events per second. Despite the advantages that the FPGA approaches offer compared to ASICs in terms of hardware reconfigurability, there is still a gap regarding the power consumption and total area required for the same design [Kuon and Rose, 2007].

A different approach which has been rapidly gaining popularity is the simulation of SNNs on general-purpose graphics processing units (GPGPUs). One example is NeMo [Fidjeland et al., 2009] which has been designed to simulate up to 40 thousand Izhikevich neurons in real-time with 40 million static delta-current synapses and a peak of 400 million synaptic events per second. Moreover, in a recent study NeMo has been extended to include spike-timing dependent plasticity (STDP) [Fidjeland and Shanahan, 2010]. While GPUs are excellent platforms for parallel computation, their memory access bandwidth is a bottleneck. For very large-scale real-time simulations of SNNs on general programmable platforms it is typically not the computational cost, but the system communications that is the prime limiting factor [Moore et al., 2012; Yudanov and Reznik, 2012; Brette and Goodman, 2012].

Benchmarking power figures for neurally-inspired hardware is challenging due to the specificity of different architectures and of models simulated on them. In this chapter characteristic power metrics for the SpiNNaker platform have been identified, so they may be used to calculate the power needed to solve neural equations for diverse neuron and synaptic models, and the energy required per synaptic event. Whilst not achieving the power efficiency of dedicated neuromorphic silicon, the SpiNNaker architecture provides an excellent trade-off in terms of scalability and reconfigurability and in its extensive interconnectivity. Table 4.13 aims to provide a summary of the different platforms available for simulating SNNs mentioned in this thesis.

The methods presented in this chapter, such as the CPU utilisation monitoring method, described in section 4.1.3, and the methodology employed by the monitor core to investigate and report the status of the application cores (section 4.3.5), by polling periodically a shared memory location, are now part of the current SpiNNaker software tools. The SSM2 was being ported to the new software tools during the writing of this thesis.

Table 4.13: Summary of the hardware platforms for simulating SNNs.

	System	Neuron/Synapse Model	Synaptic Plasticity	Precision	Simulation Performance	Latencies	Energy Usage	Peak Power
SpiNNaker [Stromatias et al., 2013]	Digital	Programmable neuron/synapse, Axonal delays	Programmable [Galluppi et al., 2014b]	Programmable	Real-time, Flexible time resolution	280 ns (per hop)	8 nJ/SE	>1 W (per chip)
TrueNorth [Merolla et al., 2014b]	Digital	Fixed neuron model w configurable params, delta-current synapse, Axonal delays	Not supported	256×410 bits per neurosynaptic core for params. & states, 4-bit synapses	Real-time, or up to 5 × faster than real-time	Not reported	26 pJ/SE	72 mW (per chip)
Neurogrid [Benjamin et al., 2014]	Mixed-mode	Fixed models, configurable params	Fixed rule	13-bit shared synapses	Real-time	1 us	941 pJ/SE	>1 W (per chip)
Hi-CANN [Schemmel et al., 2010]	Mixed-mode	Fixed models, Configurable params	Programmable [Friedmann et al., 2013]	4-/8-bit DAC synapses	Faster than real-time (up to 10^5)	4 ns (per hop)	7.41 nJ/SE (network only)	>1 kW (per wafer)
HiAER-IFAT [Yu et al., 2012]	Mixed-mode	Fixed models, configurable params	Not supported	Analogue neuron/synapse	Real-time	450 us [park2012]	22 pJ/SE	252 uW
BlueHive [Moore et al., 2012]	FPGA	Programmable neuron, delta-current synapse	Not supported	Programmable	Real-time	50 ns	Not reported	Not reported
NeMo [Fidjeland et al., 2009]	GPU	Programmable neuron, delta-current synapse	Programmable [Fidjeland and Shanahan, 2010]	Programmable	Activity-dependent	Not reported	Not reported	<200 W [nvi]
Compass [Preissl et al., 2012]	Supercomputer IBM Blue Gene/Q	Programmable	Programmable	Programmable	1,542 × slower than real-time [Wong et al., 2013]	Not reported	Not reported	7.8 MW [top]

Chapter 5

Robustness of Spiking Deep Belief Networks

Increasingly large deep learning architectures, such as DBNs are the focus of current machine learning research and achieve state-of-the-art results in different domains. However, both training and execution of large-scale Deep Networks requires vast computing resources, leading to high power requirements and communication overheads. The on-going work on design and construction of spike-based hardware platforms offers an alternative for running deep neural networks with significantly lower power consumption, but has to overcome hardware limitations in terms of noise and limited weight precision, as well as noise inherent in the sensor signal. This chapter demonstrates how such hardware constraints impact the performance of spiking neural network implementations of DBNs. In particular, the influence of limited bit precision during execution and training, and the impact of silicon mismatch in the synaptic weight parameters of custom hybrid VLSI implementations is studied. Furthermore, the network performance of spiking DBNs is characterised with regard to noise in the spiking input signal.

The results of this chapter demonstrate that spiking DBNs can tolerate very low levels of hardware bit precision down to almost two bits. Moreover, this chapter introduces a realisation of spike-based variations of DBNs on the biologically-inspired parallel SpiNNaker platform. Two spiking DBN architectures are investigated; a DBN with 2 hidden layers, as published in O'Connor et al. [2013] and implemented on an FPGA by Neil and Liu [2014], and a novel DBN with 7 hidden layers. The DBNs on SpiNNaker ran in real-time and achieved a classification performance of 94.94% and 96.22%, respectively, on the MNIST handwritten digit dataset, which at the time of publication [Stromatias et al., 2015c,a,b] was the highest score on this particular dataset

using spiking DBNs. More importantly, the difference in the classification performance between SpiNNaker and that of a pure software implementation is only 0.06%.

Using a neurally-inspired architecture yields additional benefits: during network run-time on this task, the platform consumes only 0.38 W with classification latencies in the order of tens of milliseconds, making it suitable for implementing such networks on a mobile platform. The results of this chapter also estimate how the power dissipation of the SpiNNaker platform and the classification latency of a network scales with the number of neurons and layers in the network and the overall spike activity rate. Spiking DBNs thus present an important use-case for large-scale hybrid analogue-digital or digital neuromorphic platforms such as SpiNNaker, which can execute large but precision-constrained deep networks in real time.

This chapter reproduces the author's work submitted to the Frontiers in neuromorphic engineering Journal, the 2015 International Joint Conference on Neural Networks and the 2015 International Symposium on Circuits and Systems.

5.1 Material & Methods

5.1.1 Spiking Deep Belief Networks

For this chapter the formalism for training and executing spiking DBNs developed by O'Connor et al. [2013] is being used. DBNs are multi-layered neural networks, in which each layer pair is formed by an RBM. The two layers of visible and hidden units of a RBM are fully and recurrently connected, but there are no connections between neurons of the same layer (Figure 5.1(a)). In a conventional RBM, each unit is a stochastic binary neuron, and the probability to turn on is given by a sigmoid function applied to the weighted sum of its inputs. Layers are trained one after another with an unsupervised rule called CD [Hinton and Salakhutdinov, 2006]. When training of one layer is finished, the output of the hidden units of one layer serves as the input to visible units of the subsequent layer. Supervised learning is used at the top level, where a label is jointly trained with the input, and this serves as the output of the network.

Spiking DBNs use a training procedure that is very similar to conventional DBN training, discussed in O'Connor et al. [2013], to yield the connection weights that are correct for a network of spiking neurons. Once these weights $\vec{w}$ of each RBM have been fixed, the LIF neurons follow the standard dynamics for the membrane potential

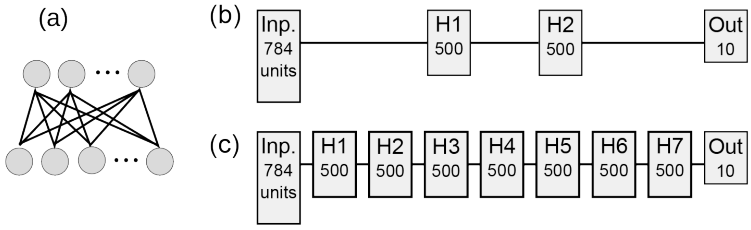


Figure 5.1: Architecture of RBMs and DBNs. (a) Architecture of a single Restricted Boltzmann Machine with full connectivity between visible units (bottom) and hidden units (top), but no connections within the same layer. (b) Topology of a DBN for MNIST classification consisting of one input layer with 784 neurons, 2 hidden layers with 500 neurons each, and a 10 neuron output layer. This is abbreviated as 784-500-500-10 architecture. (c) Topology of a DBN for MNIST classification consisting of one input layer with 784 neurons, 7 hidden layers with 500 neurons each and 10 neuron output layer. This is abbreviated as 784-500-500-500-500-500-500-500-10 architecture.

V, described as

$$\tau_m \frac{dV}{dt} = E_L - V + R_m I \quad , \tag{5.1}$$

where τ_m is the membrane time constant, E_L the resting potential, and R_m the membrane resistance. The input current I is computed as

$$I = \sum_{i=0}^{n} w_i \sum_{j=0}^{m_i} \delta(t - t_{ij}) \quad , \tag{5.2}$$

where n is the number of incoming synapses, w_i is the weight of synapse i, m_i is the number of spikes arriving at that synapse, and $\delta(t)$ is a Dirac delta function which is zero except for the firing times t_{ij} of the i^{th} input neuron. Once the membrane potential V crosses the threshold voltage V_{thresh} a spike is generated, the membrane potential is reset to V_{reset} and the neuron is not allowed to fire during the refractory period T_{refract}. Default values of the parameters used in simulations are defined in Table 5.1.

Table 5.1: Default parameters of the Leaky Integrate-and-Fire Model used in simulations.

Parameters	**Values**	**Units**
τ_m	5.0	s
T_{refract}	2.0	ms
V_{reset}	0.0	mV
V_{thresh}	1.0	mV

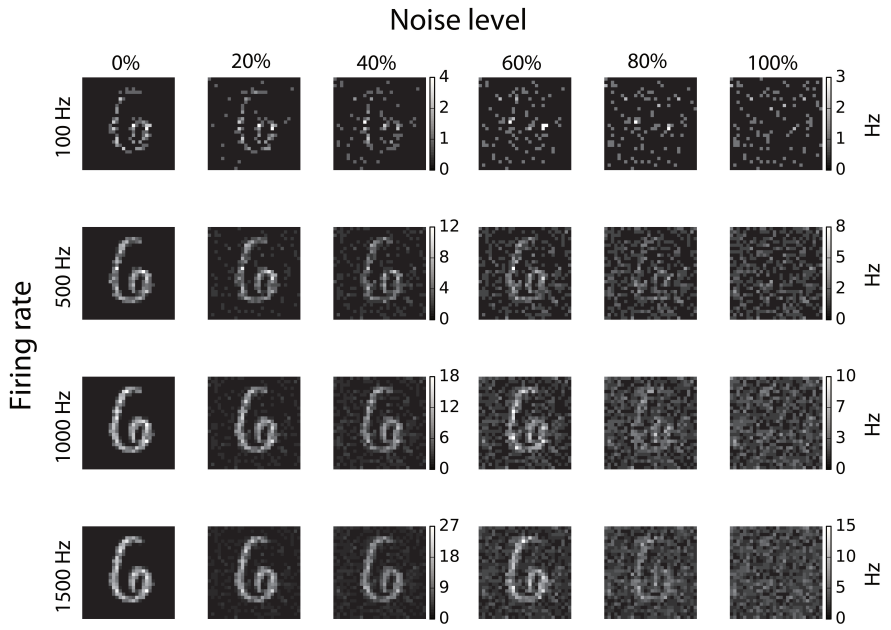


Figure 5.2: Conversion of static images to spike-trains and introduction of noise. Each row represents different input rates ranging from 100 Hz to 1500 Hz, while the columns show different percentages of input noise, from 0% up to 100%.

5.1.2 Database, Image Conversion to Spikes, and Input Noise Generation

Training and testing of spiking DBNs was carried out on the well-known MNIST database of handwritten digits [LeCun et al., 1998b], which consists of 70,000 28×28 gray-scale pixel images, of which 10,000 are used as a test set. In order to convert the static images to spike-trains, each pixel of an MNIST image is converted to a Poisson spike-train with a rate proportional to its intensity, while all firing rates are scaled such that the total firing rate of the population is constant [O'Connor et al., 2013].

To determine the impact of input noise on the performance of DBNs, noise is introduced into the spike-train representation of each image by redistributing a percentage of spikes randomly across the whole input population [Neil and Liu, 2014]. The resulting digits with different noise levels are shown in Figure 5.2, where each column represents different levels of noise starting from 0% redistribution in the first column, to 100% in the last column.

5.1.3 Conversion From Double to Lower Precision Representations

To simulate spiking DBNs implemented on digital architectures with limited hardware resources [Moore et al., 2012; Merolla et al., 2014b; Neil and Liu, 2014; Furber et al., 2014], the weights learned with double floating-point precision during training are converted to lower bit-precision representations.

Throughout this chapter the Q*m.f* notation is used to indicate a fixed-point format where m is the number of bits in the integer part, including the sign bit, followed by a notional binary point, and f is the number of bits in the fractional part. This format is a bit-level format for storing a numeric value.

Contrary to the fixed-point format, the double-precision floating-point numbers according to the IEEE 754 standard have a 64-bit word length of which 52 bits are used to store for the fraction, 11 bits for the exponent and 1 bit for the sign. In addition, a floating-point unit (FPU) is needed for computations with floating-point numbers, which results in increased area of the hardware design and higher energy costs.

The bit precision of the synaptic weights was set by keeping the number of integer bits constant to a value that is capable of holding the maximum and minimum weight value of a particular DBN, while the number of bits in the fractional part is varied from eight bits down to one bit. The double precision floating-point weights W_H are converted to lower-precision representation W_L using the conversion

$$W_L = \text{round}(2^f \cdot W_H) \cdot 2^{-f} \tag{5.3}$$

where W_H are the original double floating-point weight values of the trained DBN, and 2^{-f} is the resolution of the lower precision representation.

5.1.4 Introducing Weight Variability

To investigate the effect of mismatch when mapping the spiking DBN to mixed-mode analogue/digital multi-neuron transistor circuits, a particular synaptic circuit is considered known as the digital-to-analogue converter (DAC) synapse [Serrano-Gotarredona et al., 2008; Wang and Liu, 2013]. In this circuit, the synaptic weight is represented as a current. This synapse has a maximum current and the number of bits in the DAC sets the resolution of the synaptic current (weight). In considering the effect of mismatch due to silicon fabrication, the assumption made is that the maximum current of each DAC synapse is sampled from a Gaussian distribution. The variability is controlled

by defining the coefficient of variation (CV) of the distribution from which weights are sampled, ranging from 10% to 40%. In this analysis, calibration using the DAC to account for the mismatch in the maximum current is not included. Calibrating the DAC synapse to reduce the transistor mismatch even over the network of the size used in this work would require a long testing time.

5.1.5 Mapping spiking DBNs to SpiNNaker

Spiking DBNs were recently implemented on portable digital hardware platforms and neurally inspired hardware, which dissipate much less power than conventional processors. In particular, this has been shown by an FPGA implementation, dubbed Minitaur [Neil and Liu, 2014], and on the SpiNNaker spiking neural platform [Stromatias et al., 2015b,a]. The software simulation results in this chapter are validated by comparing to spiking DBNs running on the SpiNNaker platform.

To implement spiking DBNs on SpiNNaker, a collection of functions were developed in Python that read a MATLAB file from an off-line trained DBN [O'Connor et al., 2013] and automatically generate a PyNN [Davison et al., 2009] description of the network ready to run on SpiNNaker. The same network can also be tested on the Brian [Goodman and Brette, 2008] spiking neural network simulator as a method to verify the classification performance of spiking DBNs on SpiNNaker. The PyNN description of a spiking DBN can then easily be mapped onto the SpiNNaker platform by utilising a tool named PACMAN [Galluppi et al., 2012d].

For the input population of a spiking DBN the spike-trains generated from an MNIST digit are described as spike arrays in PyNN using the *SpikeSourceArray* population. The author developed the code that converts the spikes of a *SpikeSourceArray* to a binary file which gets uploaded to a SpiNNaker machine. A *SpikeSourceArray* kernel on an ARM9 core in a SpiNNaker chip fetches a portion of the shared memory at every millisecond and checks which bits are set in order to generate an MC packet (spike) with the appropriate neuron ID.

5.2 Experimental Results

The performance of spiking DBNs with reduced precision or weight variability is assessed on a common benchmark task, the classification of handwritten digits from the MNIST dataset [LeCun et al., 1998b]. The MNIST dataset is divided into a 60,000

digit training set, and a 10,000 digit test set. The conversion of the 28×28 gray-scale pixel images into spike trains and training of DBNs were described in Section 5.1.2. Simulation results of the spike-based DBN were obtained using the Brian spiking neural network simulator [Goodman and Brette, 2008].

In the following subsections the impact of reduced bit precision and noise on the classification performance of spiking DBNs is characterised. In Section 5.2.1 the impact of lower precision or higher input noise levels on fully trained networks during testing is demonstrated, along with an investigation of the possible reasons for the resulting performance curves in Section 5.2.2. Section 5.2.3 shows how variance on the weight parameters due to transistor mismatch affects the network performance for a particular analogue synaptic circuit. These results are verified by a comparison between the performance of a software simulation in Brian versus that of the corresponding implementation on the hardware SpiNNaker platform (Section 5.2.4). Finally, Section 5.2.5 presents a spiking DBN running on SpiNNaker in real-time with an event-driven dynamic vision sensor (DVS) [Leñero-Bardallo et al., 2011] as its input layer.

5.2.1 Robustness to reduced bit precision of fixed-point synapses

Reduction in bit precision will reduce the resources needed on a digital chip [Underwood, 2004]. If the performance of the network is maintained even when the bit precision drops, then a larger network can be implemented for the same amount of resources. The impact of the bit precision on the trained double precision floating-point weights of the DBN with the 2 hidden layers (Figure 5.1(b)) can be seen in Figure 5.3(a). Shown in the figure are the receptive fields of six of the neurons in the first hidden layer (Layer 1) for different fixed-point precisions of the synapses, ranging from double precision in the first column, to weights down to one bit for the fractional part in the last column. The figure shows that a lot of the structure in the receptive fields is still retained even with a bit precision of down to f=4 bits. Figure 5.3 (b) shows the percentage of synapses that were set to zero due to the bit reduction in the fractional part. Most compelling is that even at Q3.4, almost 50% of the weights are zero, which means these synapses are obsolete and can be pruned, thereby reducing the necessary resources even further. A similar effect can be observed for the DBN with the 7 hidden layers (Figure 5.1(c)) in Figures 5.3(c) & (d). This time when the weight precision is set to Q4.3, 50% of the weights are zero and can be pruned.

Figure 5.4(a) shows the classification accuracy (CA) of the spike-based DBN with

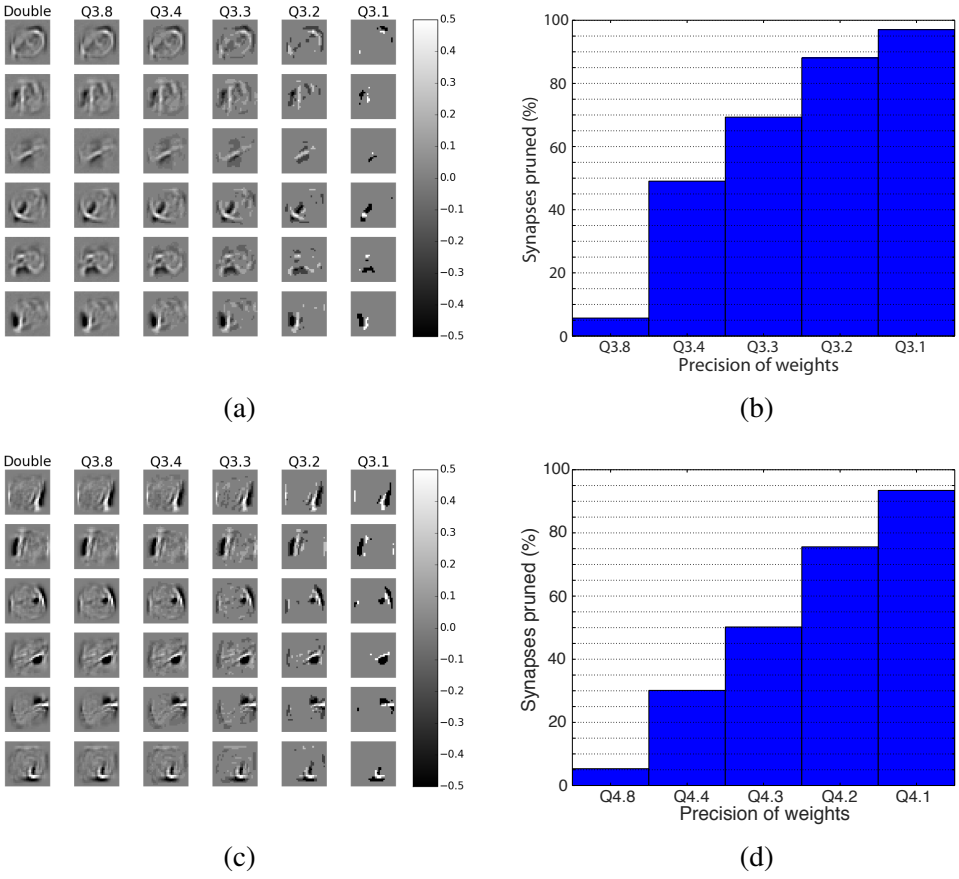


Figure 5.3: Impact of weight bit precision on the representations within a DBN. (a) The receptive fields of the first six neurons (rows) in the first hidden layer of the DBN with the 2 hidden layers and (c) the DBN with 7 hidden layers. (b) Percentage of synapses from all layers that are set to zero due to the reduction in bit precision for the fractional part for the DBN with 2 hidden layers and (d) for the DBN with 7 hidden layers.

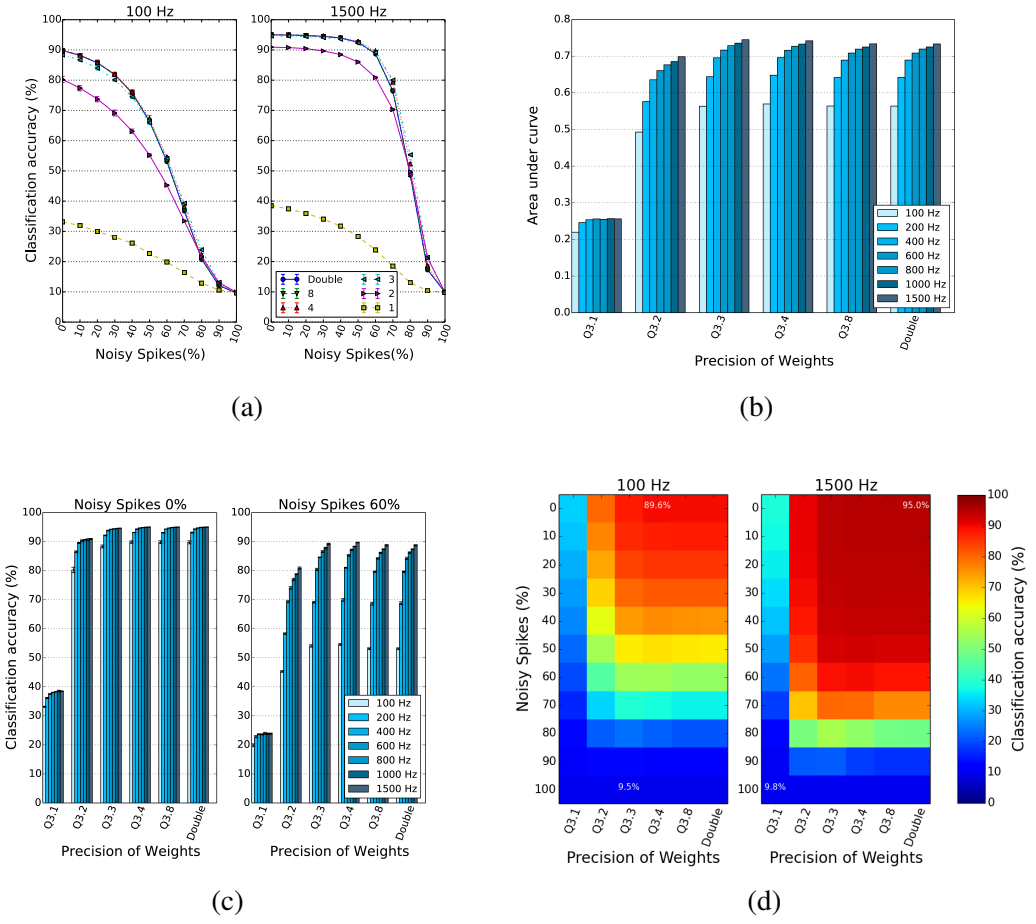
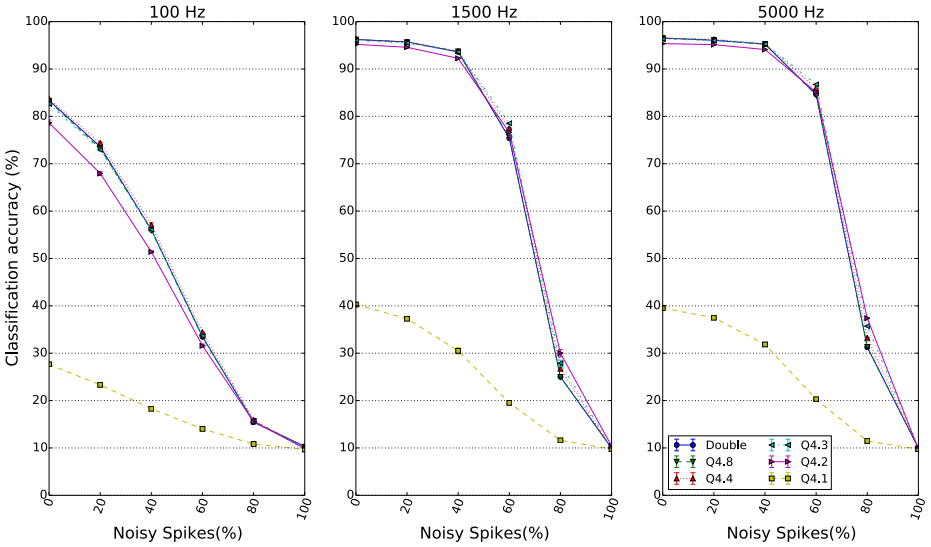
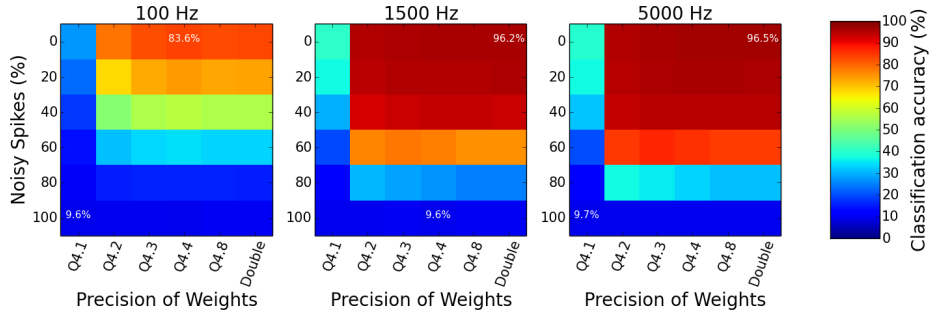


Figure 5.4: Effect of reduced weight bit precision and input noise on the classification accuracy (CA) of the spiking DBN with the 2 hidden layers. (a) CA as a function of input noise and bit precision of synaptic weights for two specific input spike rates of 100 and 1500 Hz. Results over 4 trials. (b) Normalised area under curve in (a) for different percentages of input noise, input firing rates and weight bit precision. Higher values mean higher accuracy and better robustness to noise. (c) CA as a function of the weight bit resolution for different input firing rates and for two different noise levels, 0% and 60%. (d) CA as a 2D function of the bit resolution of the weights and the percentage of input noise for 100Hz and 1500Hz input rate. The results confirm that spiking DBNs with low precision weights down to $f = 3$ bits can still reach high performance levels and tolerate high levels of input noise.



(a)



(b)

Figure 5.5: Effect of reduced weight bit precision and input noise on the classification accuracy (CA) of the spiking DBN with the 7 hidden layers. (a) CA as a function of input noise and bit precision of synaptic weights for three specific input spike rates of 100, 1500 Hz and 5000 Hz. (b) CA as a 2D function of the bit resolution of the weights and the percentage of input noise for 100, 1500 and 5000 Hz input rate. Results over 4 trials.

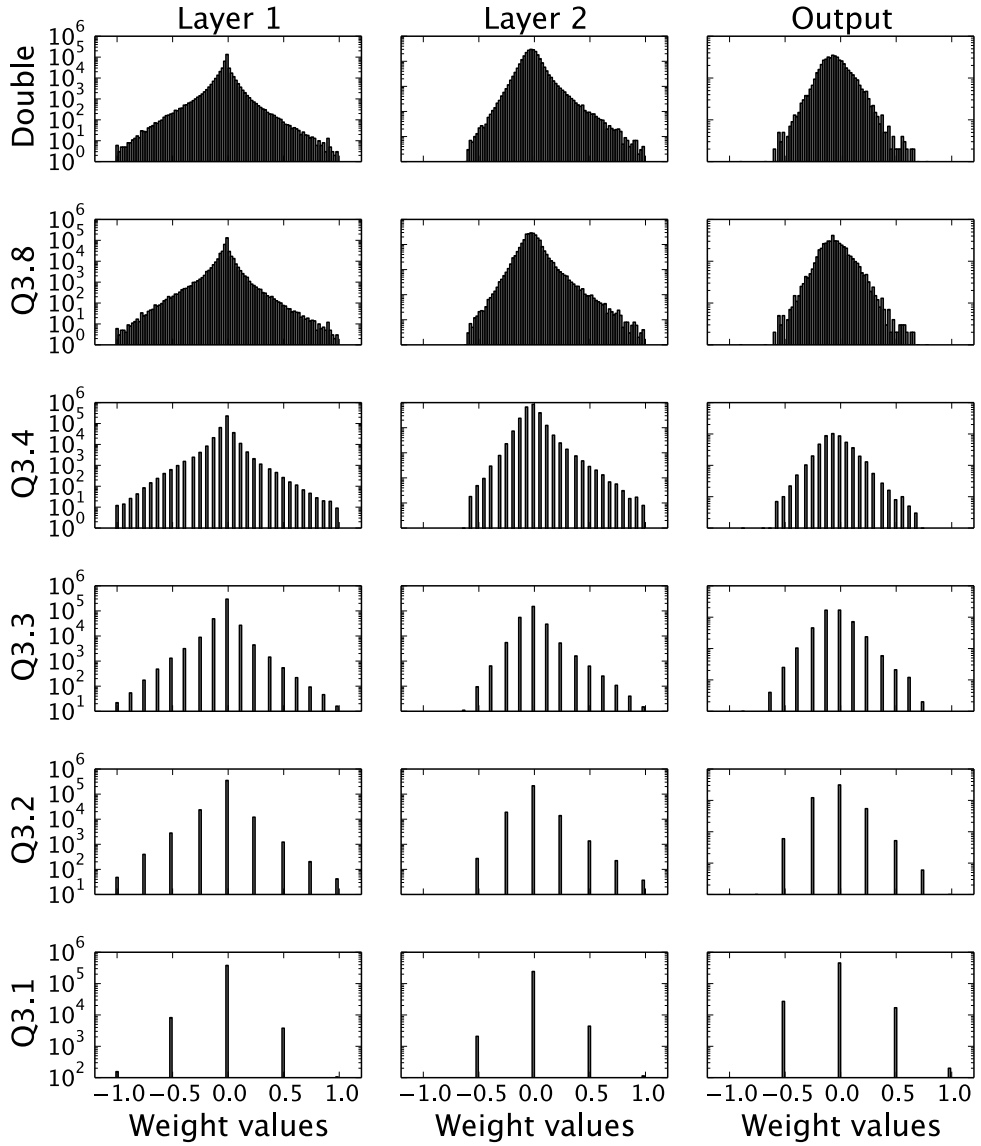
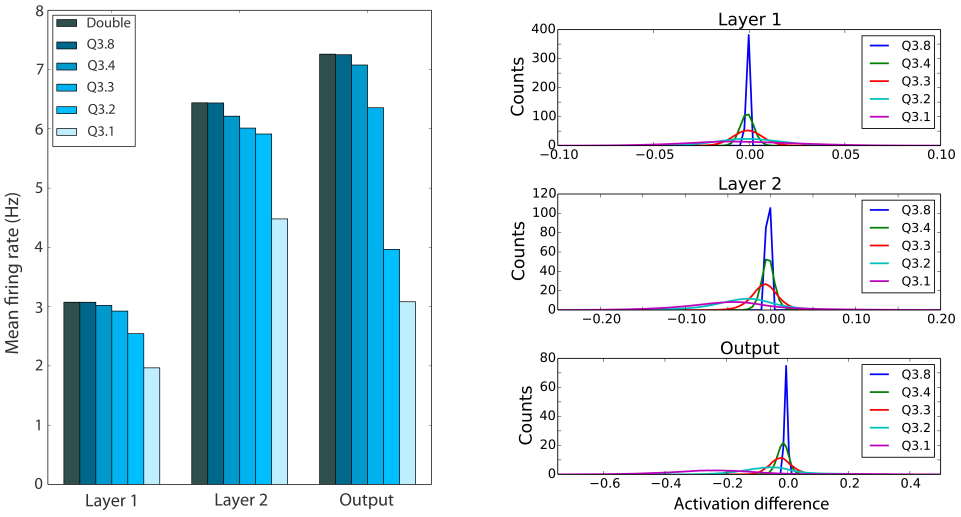


Figure 5.6: Weight distributions for different bit precision levels and DBN layers. Each row represents different fixed-point weight precisions, while each column represents a layer of the DBN, starting from Layer 1 (left), Layer 2 (middle) to the Output Layer (right). Despite the different discretization levels, the overall shape of the weight distribution is conserved.

the 2 hidden layers for different percentages of noise in the input spikes and over different input rates. The different curves within the two sets of plots in (a) show the CA as a function of the percentage of input noise spikes. The two panels show results for two different input rates, 100 Hz and 1500 Hz, which represent the total number of input spikes in the stimulus duration of 1 s. For both firing rates, the CA curves drop as the percentage of input noise spikes increases, but for 1500 Hz input the performance stays almost constant until input noise levels of 50% are reached. The different curves show the behaviour for different bit precisions of the weights. The peak performance (without noise), as well as the CA for higher input noise levels stays remarkably constant for bit precisions as low as $f = 3$. In general, reduced precision does affect the CA performance, but the peak CA value obtained for double bit precision weights decreases only by around 5% (from 95% to 90%), even when the bit precision drops to $f = 2$. In order to summarise the noise robustness for different precisions and firing rates, the area under the curve in (b) is computed, since larger area indicates both high peak performance and a slow drop-off in performance. Figure 5.4(b) shows the area under the curves in (a) as a function of the input firing rate and across 5 different bit precision values. The results show similar trends for different bit precision levels, and a similar increase in performance and noise tolerance for higher input firing rates. This is also illustrated in Figure 5.4(c), where it can be seen how the CA for different bit precisions changes as the input rates are increased from 100 Hz up to 1500 Hz, and for two different input noise levels. A drop-off in CA of around 5% to 10% for the same input rate can be observed for 60% noise spikes. The 2D plots in Figure 5.4(d) finally illustrate that there is a large range of input noise levels and bit precisions at which high performance for two different input rates can be reached. In particular, the results show that surprisingly the performance and noise robustness curves are almost identical for bit precisions down to $f = 3$ bits in all subplots. Even a synaptic weight in Q3.2 representation, which requires less than 10% of the memory resources for double precision weights gives a reasonable peak CA of 91% for low noise levels. In all subplots (a) to (d), only the Q3.1 representation shows a dramatic drop in performance.

Figure 5.5(a) presents the CA of the DBN with the 7 hidden layers for different input rates and for different percentages of noise in the input spikes. At 1500 Hz and with 0% input noise the DBN achieves a CA of 96.23%, which drops by 1% when $f = 2$. The curves show a similar trend to the ones of Figure 5.4(a) with the main difference being that the performance of the DBN with 7 hidden layers is worse than the DBN with the 2 hidden layers when the input noise reaches 60%. At 60% input



(a) (b)

Figure 5.7: Effect of reduced bit precision on firing rates in the DBN and neuron activations. (a) Mean firing rate of each layer of the network for weights with different bit precisions, using an input rate of 1500 Hz. Lower precision levels, which lead to more weights at zero, cause lower firing rates within the network. (b) Distribution of the mean difference between the activation of a neuron with double precision weights and neurons using weights with different bit precision levels. Shown are distributions over all test samples. The difference, although peaked near zero, increases for higher layers, and shows a trend towards reduced activations.

noise the CA drops to 75.4% compared to 88.7% of the DBN with the 2 hidden layers (Figure 5.4(a)). When the input rate is increased to 5000 Hz there is an approximately 9% increase in the CA for an input noise of 60%, while at 0% noise the CA rises to 96.5%. Firing rates beyond 5000 Hz have no effect on the CA or on the robustness to input noise. This increase in robustness to noise when the input rate is 5000 Hz can be also seen in Figure 5.5(b). What is remarkable is that even when the weights are set to $f = 2$ the network shows the same performance and robustness to noise as the double precision weights. The difference in robustness between the two spiking DBN topologies is probably due to the different training parameters (refer to Appendix C.1) and further investigation is required.

5.2.2 Distribution of reduced bit precision weights

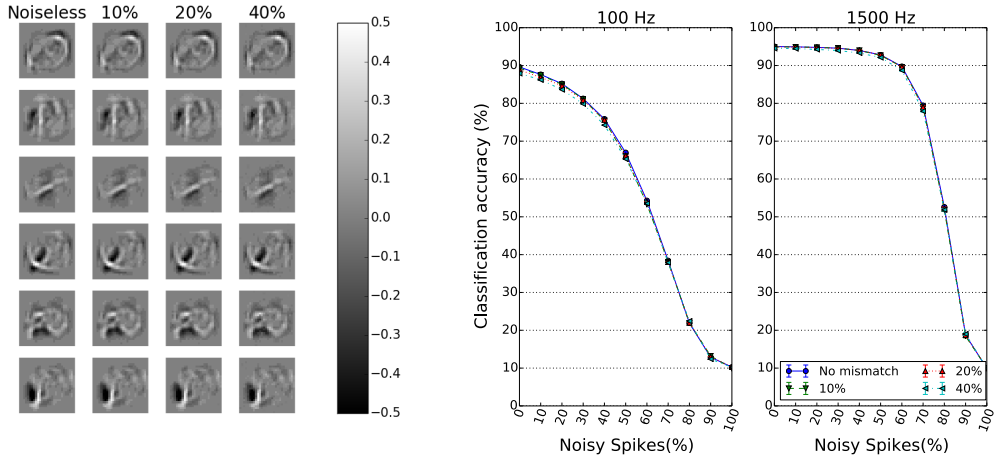
In order to understand better the surprising tolerance of spiking DBNs to reduced weight precision and high input noise levels, the impact of the reduction of precision on the distribution of weights, neuron activations, and firing rates in the network is investigated.

There are a few tools that can be employed to investigate how the distribution of the reduced bit precision weights nonetheless manages to maintain a substantial amount of the network's classification performance. Firstly, the initial question is to investigate whether this reduction in bit precision qualitatively maintains the same weight distribution as the original. Figure 5.6 shows that the quasi-continuous distribution of weights obtained for double-precision becomes increasingly discretized as the precision f decreases. In the extreme case of a Q3.1 representation, the weight values are quantized to $\pm 0.5, \pm 1$, and 0, but nonetheless seem to reflect the shape of the original distribution.

However, even with these similar shaped weight distributions, neurons' output firing rates may become dramatically altered by the subtle coercion of weights to become more similar to each other. For this, refer to Figure 5.7(a) which shows that for even high levels of quantization, the mean output spike rate per neuron for each of the 3 layers remains quite constant down to Q3.3, before a clear drop in the mean firing rate is observed. This trend is seen for all 3 layers, but is stronger in higher layers.

Finally, since these firing rates are approximately the same, we investigated whether the net activations of the neurons for the same inputs remain similar despite the quantized weight structure. Since the net activations are sums over large numbers of synapses, any rounding effects could just average out, which would help to explain the maintenance of performance with lower precision weights observed in Figure 5.4. To investigate this proposal, Figure 5.7b plots the distribution of mean differences in the net activation between neurons with double precision weights and lower precision weights for neurons in different layers. Note how in all layers the net difference in activation is much smaller than the full range of firing rates (-1 to 1), and though the width of the distribution increases as the accuracy drops, most of the weight of this histogram is concentrated around zero difference in activation. This does imply that most neurons end up with approximately the same input activation under the quantized weights, and suggest that indeed the rounding differences tend to cancel out their effects.

Similar trends can be observed for the spiking DBN with the 7 hidden layers. Figure C.4 shows the mean difference in the net activation per layer of the 784-500-500-500-500-500-500-500-10 DBN for different weight precision schemes. Similarly to the



(a) (b)

Figure 5.8: Effect of Gaussian weight variance on the performance of spiking DBNs with the 2 hidden layers. (a) Receptive fields of 6 representative neurons in the first hidden layer after perturbation with Gaussian weight variance of different CVs. (b) Impact of Gaussian weight variance on classification accuracy. The performance is plotted as a function of input noise levels for two different input rates and different weight distribution CVs. Despite the high weight variance, the performance stays high and remains robust to input noise. All weights are set by 5 bit DAC synapses (one bit is the sign bit). Results over 4 trials.

DBN with 2 hidden layers, lower bit precisions cause the mean of the distribution to shift to the negative side.

5.2.3 Robustness to variance of synaptic weights

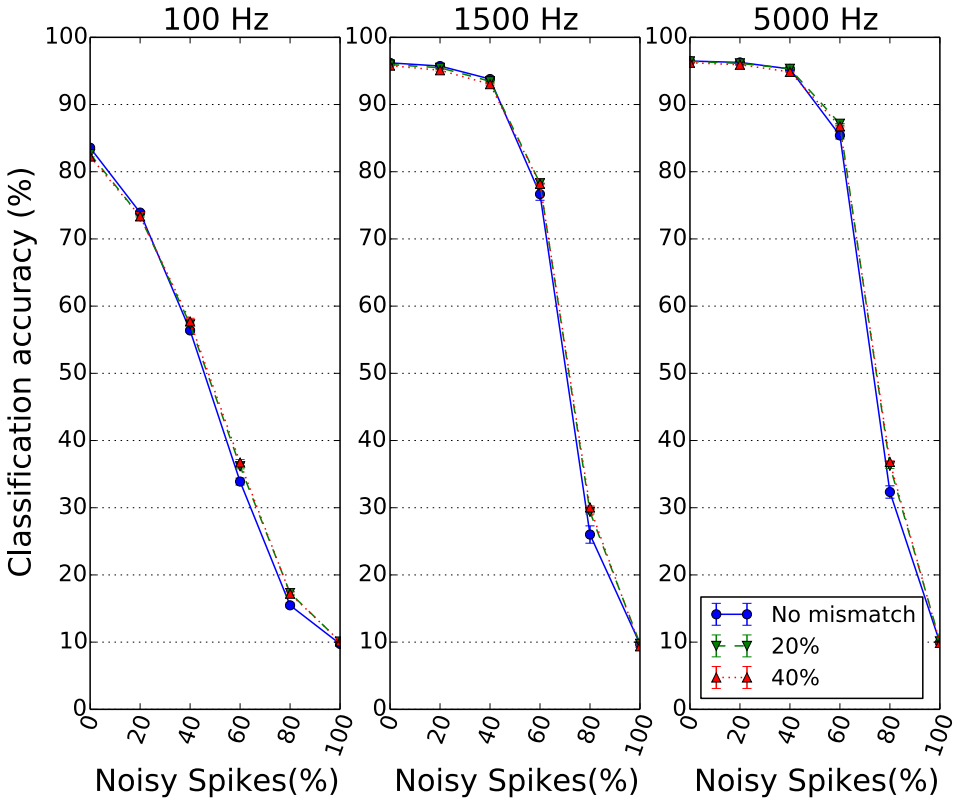
If spiking DBNs were to be implemented in analogue circuits, they have to be robust to mismatch due to the fabrication process of transistors. This process causes random variations of physical quantities (for e.g. currents) of equally sized devices and comes from sources such as the random variations in the threshold, width and length of the transistor during fabrication [Pelgrom et al., 1989; Kinget, 2005]. Measurements of these random variations is a standard practice for all silicon process technologies and is indicated by the measured standard deviation assuming a Gaussian distribution of transistor currents. Mismatch can become a factor that makes the performance of a hardware network very different from a digital simulation, and needs to be taken into account when designing mixed-mode neuron and synaptic circuits in analogue/digital multi-neuron chips [Serrano-Gotarredona et al., 2008; Wang and Liu, 2013; Brink

et al., 2013; Moradi and Indiveri, 2014]. The influence of fabrication variance is also a concern for circuits that use memristive technology [Alibart et al., 2012], where the coefficient of variation (COV) of the devices can exceed 40% in resistive values for academic technologies. The dependence of the mismatch transistor current variance on the transistor area and the current magnitude has been quantified in a CMOS 0.35 μm process [Serrano-Gotarredona and Linares-Barranco, 1999]. In order to implement the maximal number of neurons and synapses possible per chip area means that a circuit with as few transistors as possible is needed to implement their functionalities, and transistors should be small-sized. Unfortunately, the latter can lead to very large COV ($>$100%).

In order to understand the effect of parameter variance in analogue circuits on the performance of spiking DBNs, simulations were performed where synaptic weights were randomly perturbed according to the mismatch model for a particular analogue synaptic circuit. In this analysis, we chose the digital-to-analogue converter (DAC) synapse used on various neural chip implementations [Wang and Liu, 2006; Vogelstein et al., 2007; Schemmel et al., 2010; Linares-Barranco et al., 2003; Wang and Liu, 2011; Moradi and Indiveri, 2014]. The number of bits in the DAC synapse is equivalent to the f value in the Q$m.f$ format used for the bit precision. In this case, the quantized weight levels available are $I_{ref}/2^{-f}$ where I_{ref} is the maximum current that is equivalent to the maximum synaptic weight.

Mismatch measurements from 50 copies of a particular 5-bit current DAC circuit in Linares-Barranco et al. [2003] show a standard deviation around 7.77%. While the I_{ref} can be calibrated to minimize the effect of the mismatch, the assumption is made that there is no calibration because it would be too expensive to calibrate the many weights of a DBN network. In the case where a single DAC is used for positive and negative weights, then one bit is used as the sign bit.

Simulations were conducted on a network where each synapse has a 5-bit DAC. The maximum current I_{ref} = 1 nA and one bit is used as the sign bit. The circuit noise sources such as flicker noise and thermal noise are ignored in these simulations both because of the extensive time for such simulations and the dependence on the actual device sizes of the synapse. The mismatch of the transistor that supplies the maximum current for the DAC of a synapse is assumed to have a CV of 10% or 40%. The effect of applying a CV of 40% to the weights of the receptive fields of six representative neurons in the first layer of the DBN is shown in Figure 5.8(a). Despite this high CV, the receptive fields look very similar.



(a)

Figure 5.9: Effect of Gaussian weight variance on the spiking DBN with the 7 hidden layers. Results over 4 trials.

The robustness of a network with this DAC precision to Gaussian variance on I_{ref} is illustrated in Figure 5.8(b). The plots show again the performance as a function of input noise, and for 2 different input rates. The effect of increased Gaussian weight variance is minimal as can be seen from the different curves. As the CV increases to 40%, the classification accuracy in both the cases of 100 Hz and 1500 Hz input rates, decreases by $<$1% from the noiseless Q3.4 bit precision case. Similar trends can be observed for the DBN with the 7 hidden layers in Figure 5.9.

5.2.4 Comparison of hardware performance to simulations

5.2.4.1 Spike-Based DBNs on SpiNNaker

In order to validate the simulation results presented in Section 5.2.1 on actual hardware platforms, simulations of spiking DBNs were run on the fixed-point, event-based platform SpiNNaker. For the DBN with 2 hidden layers (Figure 5.1(b)), the double floating-point weights were truncated to a Q3.8 fixed-point representation, and the input rates used to encode the MNIST images into spike trains were set to 1500 Hz. The classification accuracy achieved by the SpiNNaker platform on the MNIST testing set, is 94.94% when the input noise is set to 0% and 88.66% when the input noise is set to 60%, as seen in Table 5.2. This is in good accordance with the noise-free simulation results obtained by [O'Connor et al., 2013], and classification accuracies for simulations on Brian [Goodman and Brette, 2008], which reach 94.95% and 88.66% respectively, for the same weight precision and input noise (Figure 5.4). This spiking DBN ran on a single SpiNNaker chip in real-time and generated an activity of less than 1 million synaptic events (SE) per second, which is well below the 36.8 million SE a SpiNNaker chip can process [Stromatias et al., 2013], as shown in the previous chapter. The results of the spiking DBN with the 7 hidden layers (Figure 5.1(c)) are summarised in Table 5.3. Similarly to the DBN with the 2 hidden layers the difference in the CA between Brian and SpiNNaker, for the same bit resolution, input noise and input firing rate (1500 Hz) is in the order of 0.01%.

These results indicate that the difference in the classification accuracy between SpiNNaker and Brian for the same bit resolution, input rates, and input noise levels is almost negligible and in the order of 0.01%. Moreover, the difference between the software simulation that utilises double floating-point weights and SpiNNaker with Q3.8 fixed-point weights is 0.06%, which is in agreement with a previous study [Stromatias et al., 2015a].

The performance of the SpiNNaker implementation was also compared against the software implementation in MATLAB and where possible against Minitaur [Neil and Liu, 2014], an FPGA event-based implementation of the identical off-line trained DBN with 2 hidden layers (Figure 5.1(b)).

Both the MATLAB implementation and the Brian simulator employ double floating-point arithmetic and achieved a CA of 96.06% and 95.00% respectively. In SpiNNaker the weights are represented using 8 bits for the fractional part (Q3.8), while Minitaur uses 11 bits (Q5.11). SpiNNaker achieved a CA of 94.94%, while Minitaur achieved

Table 5.2: Classification accuracies for hardware and software simulations with limited bit precision, two different input noise levels, and input rates of 1500 Hz. The first column shows results for a software simulation in Brian with Q3.8 precision, the second column shows results for SpiNNaker, which uses the same fixed point representation of weights. Differences in performance are almost negligible.

Input noise	Brian	SpiNNaker
0%	94.955%	94.94%
60%	88.665%	88.66%

Table 5.3: SpiNNaker vs Brian, for the DBN with 7 hidden layers with Q3.8 bit precision and an input rate of 1500 Hz.

Input noise	Brian	SpiNNaker
0%	96.21%	96.22%
60%	75.57%	75.55%

92% (see Table 5.4). The results indicate that there is only a 1% loss in performance when switching to spiking neuron models, which is in accordance with a previous study [O'Connor et al., 2013]. Furthermore, the SpiNNaker implementation with reduced weight precision achieves almost equivalent performance as a spiking software model. The difference in performance between the two hardware platforms (SpiNNaker and Minitaur) is likely due to the quantised look-up table Minitaur uses for the membrane decay and the event driven update of Minitaur.

Table 5.4: Classification accuracy (CA) of the same DBN with 2 hidden layers running on different platforms.

Simulator	**CA (%)**	**Weight Precision**	**Description**
Matlab	96.06	Double	Rate-based (Siegert)
Brian	95.00	Double	Clock-driven
O'Connor et al. [2013]	94.09	Double	?
SpiNNaker	94.94	Q3.8	Hybrid
Minitaur [Neil and Liu, 2014]	92.00	Q5.11	Event-driven

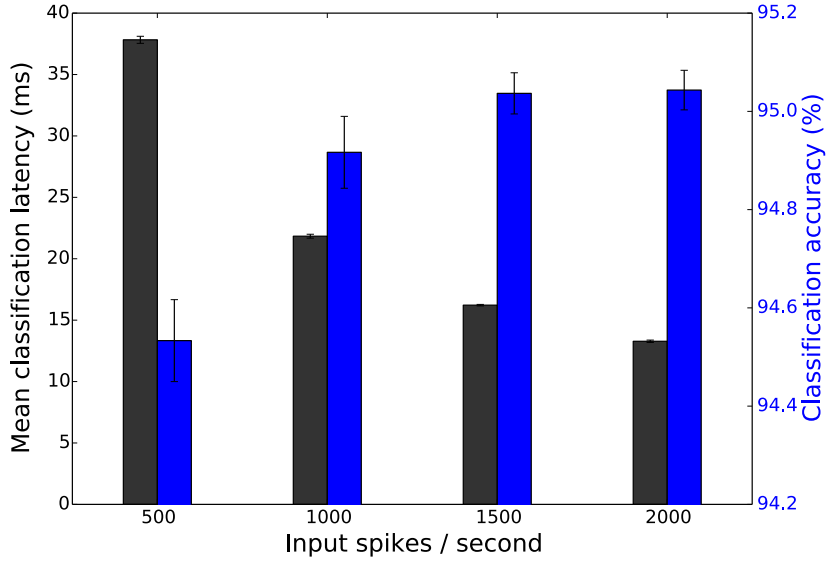
5.2.4.2 Classification Latencies

In order to investigate the real-time performance of spike-based DBNs running on SpiNNaker, two sets of experiments are conducted. The first experiment investigates how the mean classification latency and accuracy are affected by the total number of input spikes. The second experiment measures the mean classification latency of the spiking DBN as implemented on SpiNNaker.

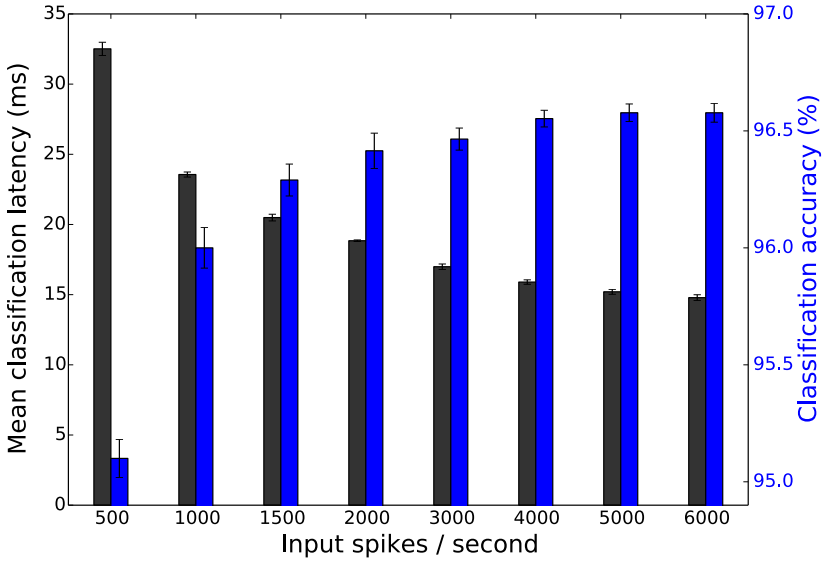
For the first experiment, the static images of the MNIST test digits are converted to spike-trains using rates from 500 Hz up to the point where additional input spikes per second have no effect on the mean classification accuracy. Each experiment ran for 4 trials and results were averaged across all trials.

For the spiking DBN with the 2 hidden layers increasing the number of input spikes reduces the mean classification latency as seen in Figure 5.10(a). At 1500 Hz the mean classification is 16.2 ms and the classification accuracy is 95.0%, while firing rates above 1500 Hz have no effect on the mean classification accuracy. Increasing the input spikes to 2000 Hz reduces the mean classification latency to 13.2 ms. Figure 5.10(b) shows the impact of the number of input rates on the mean classification latency and accuracy of the spiking DBN with the 7 hidden layers. At 1500 Hz the mean classification latency is 20.5 ms and the accuracy is 96.21%. When the number input spikes per second increase to 5000 the mean latency drops to 15.2 ms and the classification accuracy increases to 96.57%. Finally, what can also be observed from Figures 5.10(a) and (b) is that for both spiking DBNs increasing the total number of input spikes reduces the standard deviation for both the mean classification latency and classification accuracy.

To measure the classification latency of a spike-based DBN running on SpiNNaker, a Tektronix TDS 3034B oscilloscope is used to measure the time from the first input spike to the first output spike by recording the signals from the general-purpose input/output (GPIO) pins of the SpiNNaker board. The results can be seen in Figure 5.11; Figure 5.11(a) show a mean classification latency of 16 ms for the spiking DBN with the 2 hidden layers, while the DBN with the 7 hidden layers has a latency of 20.6 ms (Figure 5.11(b)). A latency in the order of ms is expected since the timer events used to solve the neuron equations for the experiments are set to 1 ms, which is the default SpiNNaker configuration.

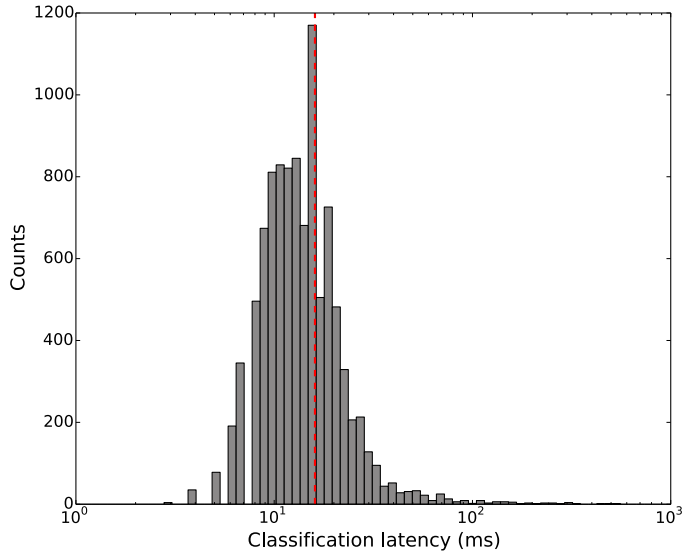


(a)

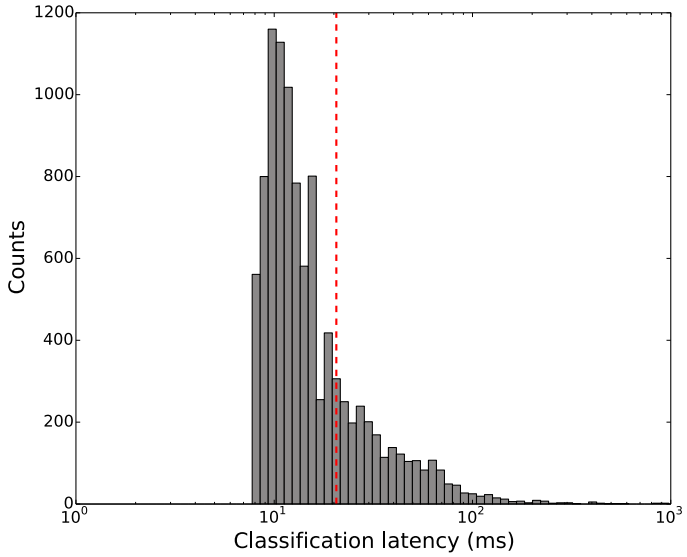


(b)

Figure 5.10: Mean classification latency and classification accuracy as a function of the input spikes per second for (a) the spiking DBN with the 2 hidden layers and for (b) the DBN with the 7 hidden layers. Results are averaged over 4 trials, error bars show standard deviations.



(a)



(b)

Figure 5.11: Histogram of the classification latencies for the MNIST digits of the testing set when the input rates are set to 1500 Hz. (a) The mean classification latency of the DBN with 2 hidden layers is 16 ms, while for the DBN with the 7 hidden layers (b) is 20.6 ms (red dashed lines). Results are from the SpiNNaker implementations.

5.2.4.3 Power Requirements of spiking DBNs on SpiNNaker

The power requirements of the SpiNNaker platform as the size of the spiking DBN scales up is explored here. The first experiment investigates the power requirements of

a spiking DBN running on a single SpiNNaker chip and then utilises a power estimation model, based on equation 4.1, to explore the scalability of spiking DBNs on larger SpiNNaker machines in terms of energy requirements.

To investigate the power requirements of a spike-based DBN running on a single SpiNNaker chip, a board with a single SpiNNaker chip was used (Figure 4.6) and the same methodology is employed as in Section 4.2.1.3. The spiking DBN with the 2 hidden layers (Figure 5.1(b)) was mapped to a SpiNNaker chip and the simulation ran for 30 seconds, while the number of input spikes generated for the same MNIST digit varied from 0 to 2000 spikes per second. Results show that both the power dissipation and number of output spikes increase with the number of input spikes per digit (Figure 5.12). When 2000 spikes per second are used per digit, a SpiNNaker chip dissipates 0.39 W, and that accounts for simulating 1794 LIF neurons with an activity of 1,569,000 synaptic events (SE) per second. For the identical spiking DBN implemented on Minitaur, which is clocked at 75 Mhz, a power dissipation of 1.5 W was reported when 1000 spikes per image were used [Neil and Liu, 2014].

A final experiment is carried out in order to investigate the power requirements of larger spiking DBNs running on a SpiNNaker board with 48 chips (Figure 4.13). The power estimation equation, Equation 4.1, is used to estimate the dependence of the power dissipation on the total number of hidden layers and neurons per hidden layer. Three different firing rates are assumed for the neurons in the hidden layer, 10 Hz, 15 Hz and 20 Hz. The results are summarised in Figure 5.13. The power estimation model is used to estimate the power under two different criteria: The minimum number of SpiNNaker chips required to simulate the total number of neurons based on the number of hidden layers and neurons per layer of the spiking DBN (150 LIF neurons per ARM9 core, 2400 per SpiNNaker chip), and the minimum amount of SpiNNaker chips required to support the total number of SE per second (36,800,000 SE per SpiNNaker chip). The total SE per second a single SpiNNaker chip can simulate in real-time were taken from Table 4.3 and it is half of what Sharp and Furber [2013] has computed. The white area in Figure 5.13 signifies the parameter regimes where real-time simulation is not feasible because the total SE per second require more than 48 SpiNNaker chips. Results show that for different topologies of spiking DBNs, the limiting factor is the number of SE as the number of hidden layers goes up, this however can be solved by using less neurons per ARM9 core. The estimated power dissipation of spiking DBNs utilising a full 48 SpiNNaker chip board is less than 32 W.

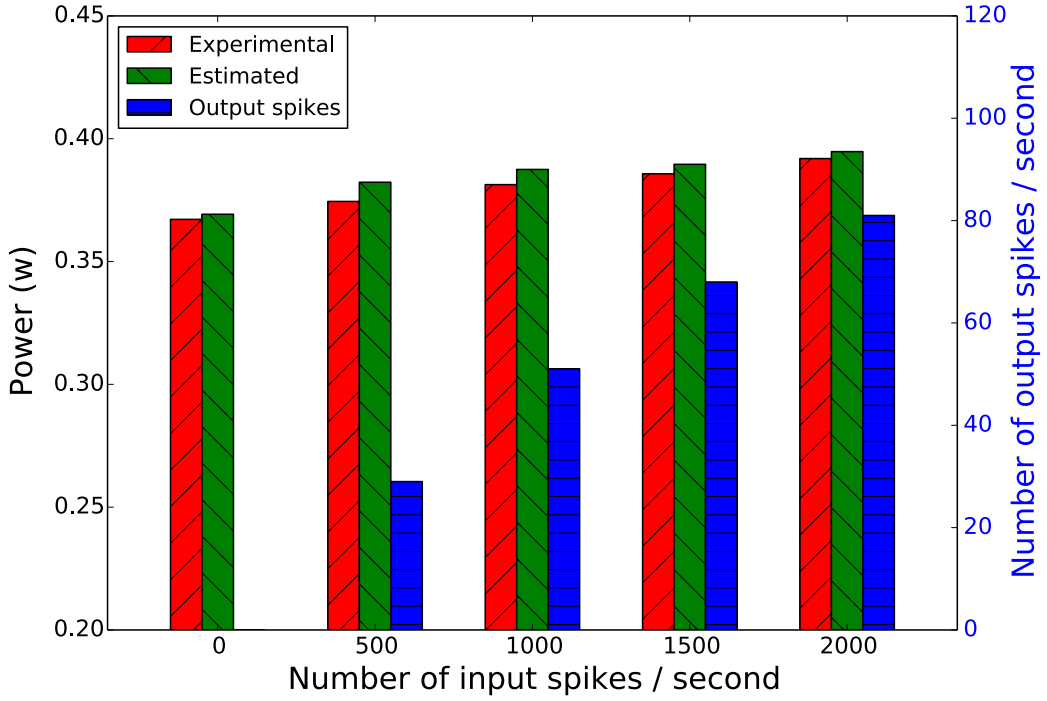


Figure 5.12: Real and estimated power dissipation of a spike-based DBN running on a single SpiNNaker chip as a function of the number of input spikes generated for the same MNIST digit. The right axis shows the number of output spikes as a function of the number of input spikes. The left bars (0 input spikes) shows power dissipation when the network is idle.

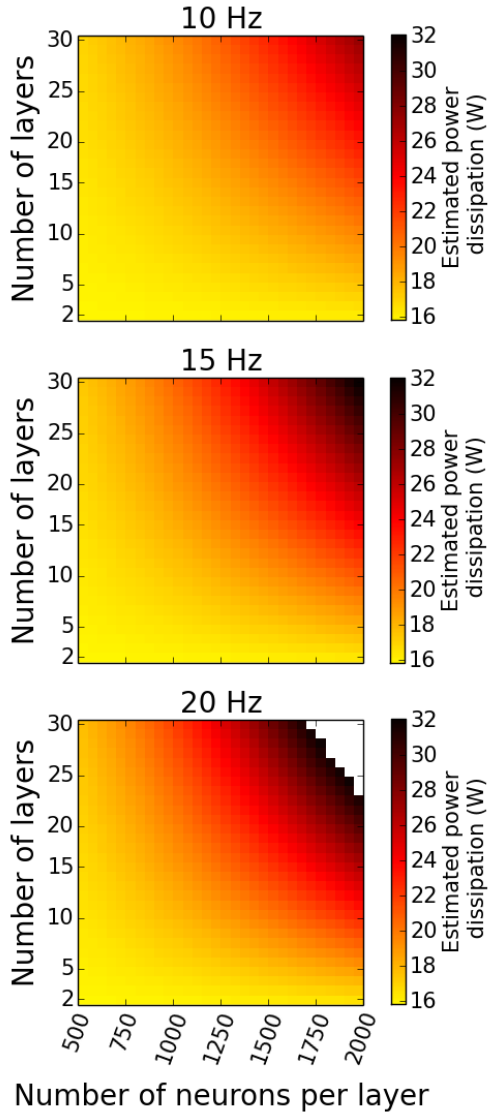


Figure 5.13: Estimating the power requirements of larger spiking DBNs running on a 48 chip SpiNNaker board, as a function of the number of hidden layers and neurons per layer, for three different firing rates for the neurons in the hidden layers. The white area denotes regimes where real-time simulation is impossible due to excessive synaptic events per second.

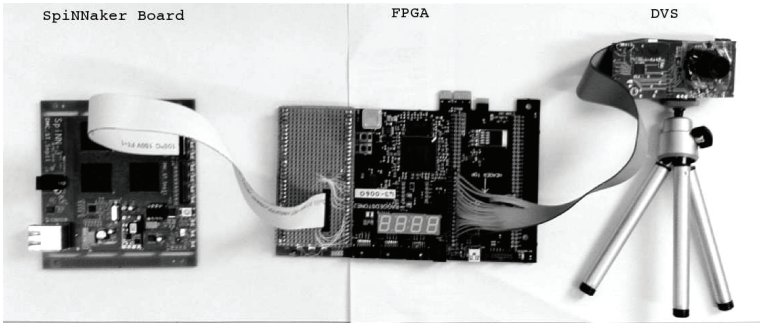


Figure 5.14: Experimental setup: A DVS silicon retina (right), an FPGA board translating the AER protocol of the DVS to a SpiNNaker compatible format (middle), a 4 SpiNNaker chip board (left) [Galluppi et al., 2012c].

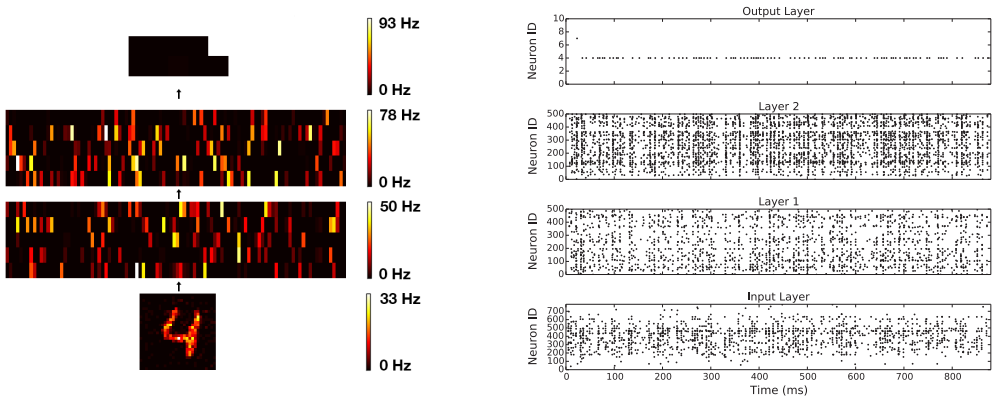
5.2.5 Neuromorphic Visual Input

This section describes the use of a dynamic vision sensor (DVS) [Leñero-Bardallo et al., 2011] as a visual input to a spiking DBN running on SpiNNaker. A DVS is an event-based image sensor comprising 128 $\times$ 128 pixels, where each pixel generates a 16-bit AER event with its address asynchronously in response to changes in the light-intensity.

Figure 5.14 presents the experimental set-up. Events from the DVS sensor are transmitted to a Xilinx SPARTAN-6 field programmable gate array (FPGA), which converts them to a SpiNNaker appropriate format and injects them to a SpiNNaker machine through one of the six asynchronous links. Based on an PyNN script, PACMAN [Galluppi et al., 2012d] is responsible for mapping the incoming MC packets (spikes), from the FPGA, to the neural network. Results are summarised in Figure 5.15. Figure 5.15(a) shows the firing rates of the spiking DBN for each layer, while Figure 5.15(b) shows a raster plot of the spiking DBN.

5.3 Summary

After outperforming other machine learning approaches on typical benchmark tasks in vision and audition, transferring Deep Learning techniques into marketable applications has become the next big target, and is supported by large ongoing industrial efforts. One of the biggest challenges is making the classification results of deep networks available in real-time, which is necessary to improve user experience for relevant applications such as speech recognition or visual object recognition. It has become clear that apart from cloud computing solutions, which require additional communication overheads,



(a) (b)

Figure 5.15: (a) The topology and firing rates of the 784-500-500-10 spiking DBN under investigation for a single input MNIST digit. The bottom plot shows firing rates of the DVS (input population). The next two rows of 5×100 show the firing rates of the neurons in the first and second hidden layer (500 neurons each), and finally the top plot shows the firing rates of the 10 neurons in the output population, one for each digit from 0 to 9. The arrows indicate all-to-all connections, which means that a neuron in one layer connects to all neurons in the next layer. (b) Raster plots of each layer of the spiking DBN.

the development of special purpose hardware to support deep networks is one of the most promising routes, in particular for mobile platforms with limited resources. Spiking deep networks have demonstrated very favourable properties in terms of latency and scaling [O'Connor et al., 2013; Stromatias et al., 2015b,a], and are a good match for ongoing efforts to advance the design of both digital and mixed-signal neuro-inspired hardware platforms [Merolla et al., 2014b; Pfeil et al., 2013; Furber et al., 2014; Pham et al., 2011; Liu et al., 2015]. Such implementations range from custom analogue mixed-signal multi-neuron platforms to more general digital platforms such as FPGAs and SpiNNaker. Thus, the present investigation of the impact of digital bit precision, input rates, and analogue transistor mismatch on the performance of the hardware implementation of a spike-based DBN is of high relevance to justify the development of larger neuromorphic platforms that support larger networks. This is particularly relevant since theory tells us that the performance of DBNs increases with the numbers of layers [Hinton and Salakhutdinov, 2006], although this does not necessarily generalize to multi-layered networks with reduced weight precision. Note that here the focus is on mapping networks that have been trained off-chip to neuromorphic hardware, rather than training networks on chip. This is because current training methods for deep networks from large datasets are optimized for exploiting conventional computing technology such as GPUs, but the execution on event-based platforms yields efficiency advantages as discussed previously.

The results show indeed that spike-based DBNs exhibit the desired robustness to input noise, and numerical precision. The classification performance of the spike-based DBN on the MNIST digit database holds up even for bit precisions down to Q3.3, which requires significantly fewer bits to represent the large parameter space of DBNs than in typical CPU systems. For example, the 2 hidden layer DBN has 642,510 synapses, which would require 4.9 MBytes if they were stored in double floating-point precision (64 bits per weight). This reduces to only 0.46 MByte, or less than 10% if weights are stored in Q3.3, i.e. 6 bit per weight precision. The DBN with the 7 hidden layers has 1,897,000 synapses which would require 15.1 MBytes to store them using double floating-point precision. If converted to Q4.2 precision then this drops to 1.4 MBytes (90.6% reduction in size). Furthermore, one of the effects of the reduced precision is that many of the weights, which typically are distributed around zero, actually become zero. For low precisions, this means that the performance can be maintained, although more than 50% of the weights become zero. Thus, these synapses are ineffective, and more than half of the weights can be ignored. This not only saves time during execution,

because of the savings in the memory lookup time for the synaptic weights in the case of a digital platform implementation, but also means that larger networks can be realized on the same hardware, because only a smaller percentage of the weights actually need to be represented. A validation was achieved by running the DBN for MNIST classification on the biologically-inspired massively-parallel fixed-point SpiNNaker platform, which uses less precise weights than standard software implementations. It has been shown that the resulting performance of the network implemented on SpiNNaker is very close to the results from the software simulation with only a 0.06% difference, despite the fact the SpiNNaker uses fixed-point arithmetic.

For implementations on custom mixed-signal hardware systems one has to deal with the constraint that they can only offer reduced numerical precision in the synaptic weights [Liu et al., 2015; Neftci et al., 2011]. The level of mismatch in the individual synapses can be taken into account during design and reduced by methods such as clever layout strategies and increasing the transistor area. Reduction of mismatch through increasing transistor area is effective [Kinget, 2005] but it increases the overall area of the synapse. Mismatch calibration methods through for example, a global Digital-to-Analogue Converter block [Oster et al., 2008] can be introduced to combat this mismatch after fabrication but the calibration itself can take a long time. The mismatch influence is also greater in low-power dissipation systems, where the transistors are usually operated in the subthreshold domain for reduced transistor current [Kinget, 2005; Linares-Barranco et al., 2003]. The results show that up to 40% of the CV for a normal distribution of mismatch can be tolerated for the network to produce approximately the same level of performance. Thus, the effects of hardware-induced imperfections seem to rather cancel out than accumulate in spiking DBNs. This study adds to current on-going studies into computational spiking network models that are robust to some level of device mismatch including that of networks with memristive devices and smaller-scale multi-neuron networks with additional spatio-temporal dynamics [Liu and Douglas, 2004; Arthur and Boahen, 2007; Vogelstein et al., 2007; Pfeil et al., 2012; Querlioz et al., 2013; Basu et al., 2013; Wang and Liu, 2013; Brink et al., 2013; Moradi and Indiveri, 2014].

Investigations of spike-based DBNs are still rare with most of the reported studies carried out on a two-layered RBM. Exceptions so far are the software DBN model in O'Connor et al. [2013], and the hardware implementation in Neil and Liu [2014]. The MNIST database was frequently used to determine the classification accuracy of the network. The software DBN of size 728-1000-500-300-50 by Eliasmith et al.

[2012] achieved 94% classification accuracy. The network used rate-based neurons except for the final output layer which was implemented with spiking neurons due to limitations on the available computing resources. Neftci et al. [2014] recently proposed an event-based variation of an online CD rule to train spiking RBMs. The trained two-layer software spike-based RBM with 824 visible neurons and 500 hidden neurons achieved a classification accuracy of 91.9%. Petrovici et al. [2013] implemented spike-based RBMs consisting of LIF neurons, following the theoretical framework of neural sampling [Buesing et al., 2011]. However, no results for the MNIST dataset are available for this approach. Arthur et al. [2012] trained a two-layer RBM consisting of 484 visible and 256 hidden units, and 10 linear classifiers in the output layer to classify the MNIST digits. The RBM was then mapped to spiking neurons by utilising a global inhibitory rhythm over fixed time windows [Merolla et al., 2010]. A hardware implementation of their digital neurosynaptic core, which contains 256 LIF neurons simulated at discrete time-steps of 1 ms, led to a classification accuracy of 89% at an energy consumption of 45 pJ. The current TrueNorth chip [Merolla et al., 2014b] consists of 4,096 such cores and has a maximum capacity of 1 million neurons, which can be simulated in real time.

In this chapter the most efficient implementation of spike-based DBNs to date was presented, running on the biologically-inspired massively-parallel fixed-point SpiNNaker platform. Its architecture is optimized for simulations with massive parallelism, asynchronous updates, and event-based chip-to-chip communication. It is an excellent fit for simulating the stereotypical neural updates and relatively sparse connections of deep networks in real-time and with minimal power consumption. Combining spiking neural networks and this hardware platform is thus an ideal fit for mobile or robotics applications [Galluppi et al., 2014a], which require fast responses while interacting with the environment, and have only a limited power budget compared to currently popular GPU- or cloud-based solutions.

The classification latencies of the implemented spiking DBNs on SpiNNaker are in the order of tens of milliseconds, which is fast enough for interactive real-time classification. Additionally, it has been demonstrated that as the number of input spikes increase, the classification accuracy improves while latency decreases. The power consumption for the spiking DBN with the 2 hidden layers running on a single SpiNNaker chip is less than 0.4 W for a digit encoded with a rate of 2000 spikes per second, while it is estimated that larger DBNs running on a larger prototype SpiNNaker board will dissipate less than 32 W.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

6.1.1 Summary

Large-scale computer simulations of ANNs have been proven to be a useful experimentation tool for computational neuroscientists, brain theoreticians and computer scientists. Computational neuroscientists aim to replicate activity observed both in in-vivo and in-vitro experiments in order to test and verify their hypotheses. Recent technological advancements allow researchers to experiment with large-scale models and results have revealed network activities and oscillations similar to in-vivo experiments [Ananthanarayanan et al., 2009; Izhikevich and Edelman, 2008]. Moreover, a large-scale functional model of the human brain, demonstrated similar response times and errors to human subjects in several cognitive tasks [Eliasmith et al., 2012].

Computer scientists on the other hand inspired from biology seek to develop new computational methods aiming to fill the gap between human and machine intelligence. Deep Learning architectures, such as Convolutional Networks and DBNs, have achieved state-of-the-art results in almost all machine learning benchmarks [Schmidhuber, 2015] and have been characterised as one of the breakthrough technologies of this decade [Hof, 2013]. Some of the advantages of DBNs is that they are scalable meaning that it has been shown to improve theoretical performance bounds by adding additional layers of neurons [Hinton and Salakhutdinov, 2006] and have the ability to learn from large unlabelled datasets [Le et al., 2012].

Simulating large neural networks is non trivial: supercomputers offer great flexibility at the price of power and communication overheads, while alternative approaches based

on GP-GPUs and FPGAs, whilst being more off-the-shelf available, show similar memory and communication bottlenecks [Moore et al., 2012; Brette and Goodman, 2012; Fatahalian and Houston, 2008]. As well as efficiency and flexibility, real-time simulation is a desirable neural network characteristic, for example in mobile and robotic platforms, which may have limited computing and power resources but require quick system responses.

The field of neuromorphic engineering aims to tackle the energy and latency issues of conventional computing systems when simulating SNNs by emulating neurons and synapses directly on hardware, efficiently and in real-time. With neuromorphic engineering it becomes possible to run large-scale simulations of SNNs in real- or accelerated-time [Schemmel et al., 2010] and perform powerful event-driven computations efficiently [Merolla et al., 2014b]. However, neuromorphic platforms suffer from model specialisation since many neuromorphic systems are highly optimised to a particular neural and synapse model and offer minimal parameter reconfigurability. If a different neuron, synapse model or plasticity rule is required then new hardware needs to be fabricated. This results in significant costs in time, human resources and finances.

The SpiNNaker platform combines the advantages of conventional computing systems and neuromorphic platforms. SpiNNaker is a fully programmable low-power system with a memory hierarchy and asynchronous communications optimised for real-time SNNs simulations. The programability of the SpiNNaker cores allow users to: develop arbitrary neuron and synapse models, implement new plasticity rules [Galluppi et al., 2014b], run simulations with heterogeneous populations of spiking neurons [Rast et al., 2011] operating at different time-scales within the same simulation [Lagorce et al., 2015], interface neuromorphic sensors [Galluppi et al., 2012c] and create closed-loop systems [Galluppi et al., 2012a, 2014a]. The communications of the system allow complex connectivity patterns while they can scale up to a billion neurons with a trillion synapses. The available software tools abstract the hardware complexity from the user making the platform accessible to non-experts.

6.1.2 Contributions

Large-scale simulations of SNNs are an important step in understanding how the brain works and to derive new computational paradigms inspired by the human brain. This is reflected by large-scale research projects and funding efforts, such as the HBP project [HBP, 2013], BRAIN initiative [Bra] and the interest shown by the industrial parties

[Sawada and Modha, 2013]. However, the power consumption of large-scale simulations on supercomputers is rarely reported and benchmarking biologically-inspired platforms is difficult due to different architectures and models simulated on them. The research presented in Chapter 4 aimed at providing a methodology to characterise: (*a*) the power requirements of a 48-node SpiNNaker board; (*b*) the communication latencies; and (*c*) to identify potential room for further optimisations. The research contribution resulted from these objectives was the development of a power estimation model based on the neuron models, number of neurons and synapses, as well as, their activity. Networks of significant sizes were executed on SpiNNaker, which at the time of publication [Stromatias et al., 2013] were the largest real-time simulation of recurrent spiking neural networks, and showed that SpiNNaker provides 54.27 million SOPS per Watt in real-time. In addition, the intra- and inter-chip core-to-core spike latencies were identified. The outcome of this research provides a basis for comparison between different neuromorphic [Merolla et al., 2014b] and biologically-inspired [Moore et al., 2012] platforms (as discussed in the summary section of Chapter 4), as well as, guidelines for further optimisations. One such optimisation was also presented in Chapter 4, where a new recoverable idle state was implemented through dynamic frequency scaling by systematically examining the energy consumption of each clocked component within a SpiNNaker chip. This proposed state showed significant power improvements, in the order of 60%, that will have greater impact on the largest SpiNNaker machine, which will utilise 1,200 48-node SpiNNaker boards, and on mobile robotic platforms that utilise SpiNNaker [Galluppi et al., 2012a, 2014a] but have a limited power budget. The results reveal that SpiNNaker, now part of the HBP, while not achieving the power efficiency of dedicated neuromorphic platforms, provides an excellent trade-off in terms of scalability and programmability.

.

A recently proposed method to map off-line trained DBNs to SNNs [O'Connor et al., 2013] paves the way towards neuromorphic accelerators able to run large-scale DBNs efficiently and with low-latency. However, since neuromorphic implementations come in various forms [Liu et al., 2015] it is necessary to investigate how their performance degrades due to hardware constraints imposed by different hardware implementations and input sensor noise. The research contribution presented in Chapter 5 is the development of a methodology to perform a full characterisation on spike-based DBNs to determine the impact of: (*a*) hardware bit precision; (*b*) analogue transistor mismatch;

(*c*) input firing rates; (*d*) input noise and (*e*) combinations of these on the classification performance of a spike-based DBNs on a handwritten digit recognition task. Results reveal that spiking DBNs can tolerate high levels of noise in the input patterns and input rates even for low weight bit resolutions. Analogue weights can tolerate up to 40% of the CV for a normal distribution of mismatch and produce approximately the same level of performance. Thus, showing that the effects of hardware-induced imperfections seem to rather cancel out than accumulate in spiking DBNs [Stromatias et al., 2015c]. These results were also validated on the SpiNNaker platform where the SpiNNaker implementation produced almost identical results, 0.06% difference in the classification performance, to Brian [Goodman and Brette, 2009] software simulator. The classification latencies of spiking DBNs on SpiNNaker were also found to be identical to Brian and are in the order of tens of milliseconds, which is fast enough for interactive real-time classification. In addition, a trade-off is observed between higher input spike rates, which require more computation but lead to better classification accuracy and lower latency, and lower spike rates which yield a more energy-efficient system. An additional contribution is that the spiking DBN implementation on SpiNNaker is the most efficient, in terms of classification accuracy and energy requirements, implementation of spike-based DBNs to date [Stromatias et al., 2015a,b]. Finally, an investigation on the scalability, in respect of power requirements, of larger spiking DBNs running on larger SpiNNaker systems utilising the power estimation model developed in Chapter 4 revealed that these type of networks will dissipate less than 32 W on a 48-node board. The studies presented in Chapter 5 intend provide important guidelines for informing current and future efforts in developing custom large-scale digital and mixed-signal spiking network platforms.

6.2 Future Work

The research described in this thesis has laid the foundations for various promising lines of future work, such as investigating the power requirements of the SpiNNaker platform when simulating purely event-driven spiking neuron models as the one developed by Lagorce et al. [2015] and will build up on the power estimation model described in Chapter 4. In addition, the experiments presented in Chapter 5 will be repeated to investigate if the event-driven neuron model has an effect on the performance of the spike-based DBNs.

A very promising feature of DBNs is their scalability: it is well known that adding

more layers to DBNs can improve performance [Hinton et al., 2006]. With the SpiNNaker architecture it becomes possible to create very large DBNs by adding additional layers, running on different cores or chips, without significantly increasing the latency of the system, and at reasonable power dissipation. Future work will thus investigate the scaling behaviour of deeper DBN architectures and other types of deep architectures on noisy or limited precision platforms. Since training remains the most computationally expensive task with DBNs, it will be interesting to study how event-based learning rules on neuromorphic platforms can contribute to speeding up this process. On-line learning rules such as the recently proposed event-based Contrastive Divergence learning rule by Neftci et al. [2014] for training a spike-based DBN can in principle utilise neuromorphic platforms for DBN training, and will have to deal with similar hardware constraints as addressed in Chapter 5. Current neuromorphic plasticity implementations are often limited to various forms of STDP, but more general plasticity frameworks such as the one recently proposed in Galluppi et al. [2014b] would provide the necessary flexibility to also test variations of contrastive divergence or related learning rules for DBNs on massively parallel brain-inspired hardware platforms.

Bibliography

The Brain Initiative. URL http://www.braininitiative.nih.gov.

nVidia Tesla C1060 Computing Processor Board, Board Specification. URL http://www.nvidia.com/docs/IO/56483/Tesla_C1060_boardSpec_v03.pdf.

TOP500 Supercomputers, SEQUOIA - BLUEGENE/Q, POWER BQC 16C 1.60 GHZ . URL http://www.top500.org/system/177556.

Neurogrid: Emulating a million neurons in the cortex. In *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, volume Supplement, pages 6702–6702, Aug 2006. doi: 10.1109/IEMBS.2006.260925.

Human Brain Projet, 2013. URL https://www.humanbrainproject.eu.

L F Abbott and S B Nelson. Synaptic plasticity: taming the beast. *Nature Neuroscience*, 3:1178–1183, 2000.

E. Adrian. *The Basis of Sensation: The Action of the Sense Organs*. W. W. Norton, New York, 1928.

Igor Aleksander. Neural systems engineering: towards a unified design discipline? *Computing Control Engineering Journal*, 1(6):259–265, Nov 1990. ISSN 0956-3385.

F. Alibart, Li. Gao, B. D. Hoskins, and D. B. Strukov. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23(7): 075201, 2012. URL http://stacks.iop.org/0957-4484/23/i=7/a=075201.

A. Amir, P. Datta, W.P. Risk, A.S. Cassidy, J.A. Kusnitz, S.K. Esser, A. Andreopoulos, T.M. Wong, M. Flickner, R. Alvarez-Icaza, E. Mcquinn, B. Shaw, N. Pass, and D.S. Modha. Cognitive computing programming paradigm: A corelet language for

composing networks of neurosynaptic cores. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–10, Aug 2013. doi: 10.1109/IJCNN.2013.6707078.

Rajagopal Ananthanarayanan, Steven K. Esser, Horst D. Simon, and Dharmendra S. Modha. The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 63:1–63:12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-744-8. doi: 10.1145/1654059.1654124. URL http://doi.acm.org/10.1145/1654059.1654124.

A. Anghel, G. Rodriguez, and B. Prisacari. The importance and characteristics of communication in high performance data analytics. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 80–81, Oct 2014. doi: 10.1109/IISWC.2014.6983044.

J.V. Arthur and K.A. Boahen. Synchrony in silicon: The gamma rhythm. *IEEE Transactions on Neural Networks*, 18(6):1815–1825, 2007. ISSN 1045-9227. doi: 10.1109/TNN.2007.900238.

J.V. Arthur, P.A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S.K. Esser, N. Imam, W. Risk, D.B.D. Rubin, R. Manohar, and D.S. Modha. Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, June 2012. doi: 10.1109/IJCNN.2012.6252637.

A. Basu, S. Shuo, H. Zhou, M. H. Lim, and G-B. Huang. Silicon spiking neurons for hardware implementation of Extreme Learning Machines. *Neurocomputing*, 102:125 – 134, 2013. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2012.01.042. URL http://www.sciencedirect.com/science/article/pii/S0925231212005814.

B.V. Benjamin, Peiran Gao, E. Mcquinn, S. Choudhary, A.R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J.V. Arthur, P.A. Merolla, and K. Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, May 2014. ISSN 0018-9219. doi: 10.1109/JPROC.2014.2313565.

GQ Bi and MM Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472, December 1998.

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 : the official journal of the Society for Neuroscience*, 2(1): 32–48, 1982. ISSN 0270-6474. doi: 10.1371/journal.ppat.0020109.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130401. URL `http://doi.acm.org/10.1145/130385.130401`.

Joseph M Brader, Walter Senn, and Stefano Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation*, 19(11): 2881–2912, 2007. ISSN 0899-7667. doi: 10.1162/neco.2007.19.11.2881.

R. Brette and W. Gerstner. Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity. *J. Neurophysiol.*, 94:3637 – 3642, 2005. ISSN 0022-3077. doi: 10.1152/jn.00686.2005. article.

Romain Brette and Dan F. M. Goodman. Simulating spiking neural networks on gpu. *Network: Computation in Neural Systems*, 23(4):167–182, 2012. doi: 10.3109/0954898X.2012.730170. URL `http://informahealthcare.com/doi/abs/10.3109/0954898X.2012.730170`. PMID: 23067314.

Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, JamesM. Bower, Markus Diesmann, Abigail Morrison, PhilipH. Goodman, Jr. Harris, FrederickC., Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, AndrewP. Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, 2007. doi: 10.1007/s10827-007-0038-6. URL `http://dx.doi.org/10.1007/s10827-007-0038-6`.

S. Brink, S. Nease, P. Hasler, S. Ramakrishnan, R. Wunderlich, A. Basu, and B. Degnan. A learning-enabled neuron array IC based upon transistor channel models of

biological phenomena. *IEEE Transactions onBiomedical Circuits and Systems*, 7(1): 71–81, Feb 2013. ISSN 1932-4545. doi: 10.1109/TBCAS.2012.2197858.

Daniel Brüderle, MihaiA. Petrovici, Bernhard Vogginger, Matthias Ehrlich, Thomas Pfeil, Sebastian Millner, Andreas Grübl, Karsten Wendt, Eric Müller, Marc-Olivier Schwartz, DanHusmann de Oliveira, Sebastian Jeltsch, Johannes Fieres, Moritz Schilling, Paul Müller, Oliver Breitwieser, Venelin Petkov, Lyle Muller, AndrewP. Davison, Pradeep Krishnamurthy, Jens Kremkow, Mikael Lundqvist, Eilif Muller, Johannes Partzsch, Stefan Scholze, Lukas Zühl, Christian Mayr, Alain Destexhe, Markus Diesmann, TobiasC. Potjans, Anders Lansner, René Schüffny, Johannes Schemmel, and Karlheinz Meier. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological Cybernetics*, 104(4-5):263–296, 2011. ISSN 0340-1200. doi: 10.1007/s00422-011-0435-9. URL http://dx.doi.org/10.1007/s00422-011-0435-9.

Nicolas Brunel. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of Computational Neuroscience*, 8(3):183–208, 2000. ISSN 0929-5313. doi: 10.1023/A:1008925309027. URL http://dx.doi.org/10.1023/A%3A1008925309027.

Lars Buesing, Johannes Bill, Bernhard Nessler, and Wolfgang Maass. Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS Computational Biology*, 7(11):e1002211, 2011.

Daniel A Butts and Mark S Goldman. Tuning curves, neuronal variability, and sensory coding. *PLoS Biology*, 4(4):e92, 04 2006. doi: 10.1371/journal.pbio.0040092. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1403159/.

L. Camuñas-Mesa, J.A. Pérez-Carrasco, C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco. On scalable spiking convnet hardware for cortex-like visual sensory processing systems. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 249–252, May 2010. doi: 10.1109/ISCAS.2010.5537918.

Stijn Cassenaer and Gilles Laurent. Conditional modulation of spike-timing-dependent plasticity for olfactory learning. *Nature*, 482(7383):47–52, 2012.

A. Cassidy, A.G. Andreou, and J. Georgiou. Design of a one million neuron single fpga neuromorphic system for real-time multimodal scene analysis. In *Information*

Sciences and Systems (CISS), 2011 45th Annual Conference on, pages 1–6, March 2011. doi: 10.1109/CISS.2011.5766099.

A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B. D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–10, Aug 2013. doi: 10.1109/IJCNN.2013.6707077.

Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011. ISBN 978-1-57735-514-4. doi: 10.5591/978-1-57735-516-8/IJCAI11-210. URL http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-210.

Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Comput.*, 22 (12):3207–3220, December 2010. ISSN 0899-7667. doi: 10.1162/NECO_a_00052. URL http://dx.doi.org/10.1162/NECO_a_00052.

Claudia Clopath, Lars Busing, Eleni Vasilaki, and Wulfram Gerstner. Connectivity reflects coding: a model of voltage-based STDP with homeostasis. *Nature Neuroscience*, 13(3):344–352, March 2010. ISSN 1097-6256. doi: 10.1038/nn.2479. URL http://dx.doi.org/10.1038/nn.2479.

Jrg Conradt, Francesco Galluppi, and Terrence C. Stewart. Trainable sensorimotor mapping in a neuromorphic robot. *Robotics and Autonomous Systems*, (0):–, 2014. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/j.robot.2014.11.004. URL http://www.sciencedirect.com/science/article/pii/S0921889014002462.

Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

G.E. Dahl, Dong Yu, Li Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language*

Processing, IEEE Transactions on, 20(1):30–42, Jan 2012. ISSN 1558-7916. doi: 10.1109/TASL.2011.2134090.

Andrew P. Davison, Daniel Brüderle, Jochen M. Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2:11, 2009.

Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005. ISBN 0262541858.

Erik De Schutter, editor. *Computational modeling methods for neuroscientists*. Computational neuroscience. Cambridge, Mass. MIT Press, 2010. ISBN 978-0-262-01327-7. URL http://opac.inria.fr/record=b1132455.

J. Dean, G. Corrado, R. Monga, M. Chen, K.and Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

P.U. Diehl and M. Cook. Efficient implementation of stdp rules on spinnaker neuromorphic hardware. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 4288–4295, July 2014. doi: 10.1109/IJCNN.2014.6889876.

C. Eliasmith and C.H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. A Bradford book. MIT Press, Cambridge, MA, USA, 2004. ISBN 9780262550604. URL http://books.google.co.uk/books?id=J6jz9s4kbfIC.

Chris Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural Computation*, 17(6):1276–1314, 2015/10/08 2005. doi: 10.1162/0899766053630332. URL http://dx.doi.org/10.1162/0899766053630332.

Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, 2012. doi: 10.1126/science.1225266.

Jochen M Eppler, Moritz Helias, Eilif Muller, Markus Diesmann, and Marc-Oliver Gewaltig. Pynest: a convenient interface to the nest simulator. *Frontiers in Neuroinformatics*, 2(12), 2009. ISSN 1662-5196. doi: 10.3389/neuro.11.012.2008. URL http://www.frontiersin.org/neuroinformatics/10.3389/neuro.11.012.2008/abstract.

H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. *Micro, IEEE*, 32(3):122 –134, may-june 2012a. ISSN 0272-1732. doi: 10.1109/MM.2012.17.

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Towards neural acceleration for general- purpose approximate computing. In *Proceedings of the 4th Workshop on Energy Efficient Design*, ISCA '12, Piscataway, NJ, USA, 2012b.

C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 109–116, 2011.

C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35 (8):1915–1929, Aug 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.231.

Kayvon Fatahalian and Mike Houston. A closer look at gpus. *Commun. ACM*, 51 (10):50–57, October 2008. ISSN 0001-0782. doi: 10.1145/1400181.1400197. URL `http://doi.acm.org/10.1145/1400181.1400197`.

A.K. Fidjeland and M.P. Shanahan. Accelerated simulation of spiking neural networks using gpus. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8, July 2010. doi: 10.1109/IJCNN.2010.5596678.

A.K. Fidjeland, E.B. Roesch, M.P. Shanahan, and W. Luk. Nemo: A platform for neural modelling of spiking neurons using gpus. In *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, pages 137–144, July 2009. doi: 10.1109/ASAP.2009.24.

Message P Forum. Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

Simon Friedmann, Nicolas Frmaux, Johannes Schemmel, Wulfram Gerstner, and Karlheinz Meier. Reward-based learning under hardware constraints - using a risc processor embedded in a neuromorphic substrate. *Frontiers in Neuroscience*, 7(160), 2013. ISSN 1662-453X. doi: 10.3389/fnins.2013.00160. URL `http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2013.00160/abstract`.

Johannes Friedrich, Robert Urbanczik, and Walter Senn. Spatio-temporal credit assignment in neuronal population learning. *PLoS Computational Biology*, 7(6):e1002092, 2011.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. ISSN 0340-1200. doi: 10.1007/BF00344251. URL http://dx.doi.org/10.1007/BF00344251.

S. Furber and A. Brown. Biologically-inspired massively-parallel architectures - computing beyond a million processors. In *Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on*, pages 3–12, July 2009. doi: 10.1109/ACSD.2009.17.

S. Furber, S. Temple, and A. Brown. On-chip and inter-chip networks for modeling large-scale neural systems. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4 pp.–, May 2006. doi: 10.1109/ISCAS.2006.1692992.

S.B. Furber, F. Galluppi, S. Temple, and L.A Plana. The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665, May 2014. ISSN 0018-9219. doi: 10.1109/JPROC.2014.2304638.

Steve Furber and Steve Temple. Neural systems engineering. *Journal of the Royal Society Interface*, 4(13):193–206, 04 2007. doi: 10.1098/rsif.2006.0177. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2359843/.

Steve B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown. Overview of the SpiNNaker system architecture. *IEEE Trans. Comput.*, 62(12):2454–2467, December 2013. ISSN 0018-9340. doi: 10.1109/TC.2012.142. URL http://dx.doi.org/10.1109/TC.2012.142.

F. Galluppi, J. Conradt, T. Stewart, C. Eliasmith, T. Horiuchi, J. Tapson, B. Tripp, S. Furber, and R. Etienne-Cummings. Live demo: Spiking ratslam: Rat hippocampus cells in spiking neural hardware. In *Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE*, pages 91–91, Nov 2012a. doi: 10.1109/BioCAS.2012.6418493.

F. Galluppi, S. Davies, S. Furber, T. Stewart, and C. Eliasmith. Real time on-chip implementation of dynamical systems with spiking neurons. In *Neural Networks*

(IJCNN), The 2012 International Joint Conference on, pages 1–8, June 2012b. doi: 10.1109/IJCNN.2012.6252706.

F. Galluppi, C. Denk, M.C. Meiner, T.C. Stewart, L.A. Plana, C. Eliasmith, S. Furber, and J. Conradt. Event-based neural computing on an autonomous mobile platform. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2862–2867, May 2014a. doi: 10.1109/ICRA.2014.6907270.

Francesco Galluppi. *Information Representation on a Universal Neural Chip*. PhD thesis, School of Computer Science, University of Manchester, 2013.

Francesco Galluppi and Steve Furber. Representing and decoding rank order codes using polychronization in a network of spiking neurons. In *The 2011 International Joint Conference on Neural Networks, IJCNN 2011, San Jose, California, USA, July 31 - August 5, 2011*, pages 943–950, 2011. doi: 10.1109/IJCNN.2011.6033324. URL http://dx.doi.org/10.1109/IJCNN.2011.6033324.

Francesco Galluppi, Kevin Brohan, Simon Davidson, Teresa Serrano-Gotarredona, José-Antonio Pérez Carrasco, Bernabé Linares-Barranco, and Steve Furber. A real-time, event-driven neuromorphic system for goal-directed attentional selection. In *Neural Information Processing*, pages 226–233. Springer, 2012c.

Francesco Galluppi, Sergio Davies, Alexander Rast, Thomas Sharp, Luis A. Plana, and Steve Furber. A hierachical configuration system for a massively parallel neural hardware platform. In *Proceedings of the 9th Conference on Computing Frontiers*, CF '12, pages 183–192, New York, NY, USA, 2012d. ACM. ISBN 978-1-4503-1215-8. doi: 10.1145/2212908.2212934. URL http://doi.acm.org/10.1145/2212908.2212934.

Francesco Galluppi, Xavier Lagorce, Evangelos Stromatias, Michael Pfeiffer, Luis A Plana, Steve B Furber, and Ryad Benjamin Benosman. A framework for plasticity implementation on the spinnaker neural architecture. *Frontiers in Neuroscience*, 8(429), 2014b. ISSN 1662-453X. doi: 10.3389/fnins.2014.00429. URL http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2014.00429/abstract.

AP Georgopoulos, AB Schwartz, and RE Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986. doi: 10.1126/science.3749885. URL http://www.sciencemag.org/content/233/4771/1416.abstract.

W Gerstner, R Kempter, J L van Hemmen, and H Wagner. A Neuronal Learning Rule for Sub-millisecond Temporal Coding. *Nature*, 383(6595):76–78, 1996.

Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

Dan F M Goodman and Romain Brette. Brian: a simulator for spiking neural networks in python. *Frontiers in Neuroinformatics*, 2(5), 2008. ISSN 1662-5196. doi: 10.3389/neuro.11.005.2008. URL http://www.frontiersin.org/neuroinformatics/10.3389/neuro.11.005.2008/abstract.

Dan F M Goodman and Romain Brette. The brian simulator. *Frontiers in Neuroscience*, 3(26), 2009. ISSN 1662-453X. doi: 10.3389/neuro.01.026.2009. URL http://www.frontiersin.org/neuroscience/10.3389/neuro.01.026.2009/abstract.

S.G. Gordon Shepherd and S. Grillner. *Handbook of Brain Microcircuits*. Oxford University Press, 2010. ISBN 9780199780334. URL https://books.google.gr/books?id=61lcGm7uhLgC.

Rudy Guyonneau, Rufin Van Rullen, and Simon J Thorpe. Neurons tune to the earliest spikes through STDP. *Neural Computation*, 17(4):859–879, 2005.

Corey B. Hart. Towards a compiler for a polychronous wavefront computer: Programming by optimization. *Procedia Computer Science*, 36:387 – 392, 2014. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2014.09.010. URL http://www.sciencedirect.com/science/article/pii/S1877050914012599. Complex Adaptive Systems Philadelphia, {PA} November 3-5, 2014.

Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, new ed edition, June 1949. ISBN 0805843000. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0805843000.

M. L. Hines and N. T. Carnevale. The neuron simulation environment. *Neural Comput.*, 9(6):1179–1209, August 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.6.1179. URL http://dx.doi.org/10.1162/neco.1997.9.6.1179.

G. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

G. Hinton, Li Deng, Dong Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, Nov 2012. ISSN 1053-5888. doi: 10.1109/MSP.2012.2205597.

G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006. doi: 10.1126/science.1127647. URL http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&uid=16873662&cmd=showdetailview&indexed=google.

A. L. Hodgkin and A. F. Huxley. Action Potentials record from inside a nerve fiber. *Nature (Lond)*, 144:710–711, October 1939.

A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117 (4):500–544, 1952. ISSN 1469-7793. doi: 10.1113/jphysiol.1952.sp004764. URL http://dx.doi.org/10.1113/jphysiol.1952.sp004764.

R. D. Hof. 10 breakthrough technologies of 2013, 2013. URL http://www.technologyreview.com/featuredstory/513696/deep-learning/.

J. J. Hopfield and Carlos D. Brody. What is a moment? "cortical" sensory integration over a brief interval. *Proceedings of the National Academy of Sciences*, 97(25): 13919–13924, 2000. doi: 10.1073/pnas.250483697. URL http://www.pnas.org/content/97/25/13919.abstract.

J. J. Hopfield and Carlos D. Brody. What is a moment? transient synchrony as a collective mechanism for spatiotemporal integration. *Proceedings of the National Academy of Sciences*, 98(3):1282–1287, 2001. doi: 10.1073/pnas.98.3.1282. URL http://www.pnas.org/content/98/3/1282.abstract.

D. H. Hubel and T. N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

Giacomo Indiveri, Bernabe Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp

Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain SAÏGHI, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5(73), 2011. ISSN 1662-453X. doi: 10.3389/fnins.2011.00073.

Intel. Online technical specification. http://ark.intel.com/products/64596/Intel-Xeon-Processor-E5-2690-20M-Cache-2_90-GHz-8_00-GTs-Intel-QPI, 2014. URL \url{http://ark.intel.com/products/64596/Intel-Xeon-Processor-E5-2690-20M-Cache-2_90-GHz-8_00-GTs-Intel-QPI}.

Intel Xeon Processor E5-2600 Series. Online technical specification. http://download.intel.com/support/processors/xeon/sb/xeon_E5-2600.pdf, 2014. URL \url{http://download.intel.com/support/processors/xeon/sb/xeon_E5-2600.pdf}.

E.M. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569 – 1572, Nov. 2003. ISSN 1045-9227. doi: 10.1109/TNN.2003.820440.

EM Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cereb Cortex*, 17(10):2443–2452, October 2007. ISSN 1047-3211. doi: 10.1093/cercor/bhl152. URL http://dx.doi.org/10.1093/cercor/bhl152http://www.ncbi.nlm.nih.gov/pubmed/17220510.

Eugene M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, September 2004. ISSN 1045-9227. doi: 10.1109/TNN.2004.832719. URL http://www.ncbi.nlm.nih.gov/pubmed/15484883.

Eugene M. Izhikevich. Polychronization: Computation with spikes. *Neural Computation*, 18(2):245–282, 2015/10/09 2006. doi: 10.1162/089976606775093882. URL http://dx.doi.org/10.1162/089976606775093882.

Eugene M Izhikevich and Gerald M Edelman. Large-scale model of mammalian thalamocortical systems. *Proc Natl Acad Sci U S A*, 105(9):3593–3598, March 2008. doi: 10.1073/pnas.0712231105. URL http://dx.doi.org/10.1073/pnas.0712231105.

Eugene M. Izhikevich and Frank C. Hoppensteadt. Polychronous wavefront computations. *I. J. Bifurcation and Chaos*, 19(5):1733–1739, 2009. URL http://dblp.uni-trier.de/db/journals/ijbc/ijbc19.html#IzhikevichH09.

H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001. URL http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf.

PhilipH.-S. Jen, Xinde Sun, and PaulJ.J. Lin. Frequency and space representation in the primary auditory cortex of the frequency modulating bateptesicus fuscus. *Journal of Comparative Physiology A*, 165(1):1–14, 1989. ISSN 0340-7594. doi: 10.1007/BF00613794. URL http://dx.doi.org/10.1007/BF00613794.

Y. Jiao, H. Lin, P. Balaji, and W. Feng. Power and performance characterization of computational kernels on the gpu. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 221–228, Dec 2010. doi: 10.1109/GreenCom-CPSCom.2010.143.

Xin Jin, S.B. Furber, and J.V. Woods. Efficient modelling of spiking neural networks on a scalable chip multiprocessor. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2812–2819, June 2008. doi: 10.1109/IJCNN.2008.4634194.

Xin Jin, Alexander Rast, Francesco Galluppi, Mukaram Khan, and Steve Furber. Implementing learning on the spinnaker universal neural chip multiprocessor. In ChiSing Leung, Minho Lee, and JonathanH. Chan, editors, *Neural Information Processing*, volume 5863 of *Lecture Notes in Computer Science*, pages 425–432. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-10676-7. doi: 10.1007/978-3-642-10677-4_48. URL http://dx.doi.org/10.1007/978-3-642-10677-4_48.

Prashant Joshi and Wolfgang Maass. Movement generation with circuits of spiking neurons. *Neural Computation*, 17(8):1715–1738, 2015/09/29 2005. doi: 10.1162/0899766054026684. URL http://dx.doi.org/10.1162/0899766054026684.

A. Joubert, B. Belhadj, O. Temam, and R. Heliot. Hardware spiking neurons design: Analog or digital? In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–5, June 2012. doi: 10.1109/IJCNN.2012.6252600.

F. Jug, M. Cook, and A. Steger. Recurrent competitive networks can learn locally excitatory topologies. In *Proceedings of 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, June 2012. doi: 10.1109/IJCNN.2012.6252786.

Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell, editors. *Principles of Neural Science*. Elsevier, New York, third edition, 1991.

R Kempter, W Gerstner, and J.˜L. van Hemmen. Intrinsic stabilization of output rates by spike-based Hebbian learning. *Neural Computation*, 13:2709–2741, 2001.

P.R. Kinget. Device mismatch and tradeoffs in the design of analog circuits. *IEEE J. Solid-State Circuits*, 40(6):1212–1224, 2005.

Eric I. Knudsen. Instructed learning in the auditory localization pathway of the barn owl. *Nature*, 417(6886):322–328, 05 2002. URL http://dx.doi.org/10.1038/417322a.

Dimitri M Kullmann, Alexandre W Moreau, Yamina Bakiri, and Elizabeth Nicholson. Plasticity of inhibition. *Neuron*, 75(6):951–962, 2012.

I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203–215, Feb 2007. ISSN 0278-0070. doi: 10.1109/TCAD.2006.884574.

Xavier Lagorce, Evangelos Stromatias, Francesco Galluppi, Luis A. Plana, Shih-Chii Liu, Steve B Furber, and Ryad Benjamin Benosman. Breaking the millisecond barrier on spinnaker: Implementing asynchronous event-based plastic models with microsecond resolution. *Frontiers in Neuroscience*, 9(206), 2015. ISSN 1662-453X. doi: 10.3389/fnins.2015.00206. URL http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2015.00206/abstract.

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 473–480, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273556. URL http://doi.acm.org/10.1145/1273496.1273556.

Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Jeffrey Dean, and Andrew Y. Ng. Building high-level features using large

scale unsupervised learning. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012. URL http://icml.cc/discuss/2012/73.html.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998a. ISSN 0018-9219. doi: 10.1109/5.726791.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998b.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 05 2015. URL http://dx.doi.org/10.1038/nature14539.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553453. URL http://doi.acm.org/10.1145/1553374.1553453.

Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, 4(10):e1000180, 2008.

J.A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco. A 3.6 μs latency asynchronous frame-free event-driven dynamic-vision-sensor. *Solid-State Circuits, IEEE Journal of*, 46(6):1443–1455, June 2011. ISSN 0018-9200. doi: 10.1109/JSSC.2011.2118490.

Bil Lewis and Daniel J. Berg. *Multithreaded Programming with Pthreads*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998. ISBN 0-13-680729-1.

P. Lichtsteiner, C. Posch, and T. Delbruck. A 128$\times$128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *Solid-State Circuits, IEEE Journal of*, 43 (2):566–576, Feb 2008. ISSN 0018-9200. doi: 10.1109/JSSC.2007.914337.

B. Linares-Barranco, T. Serrano-Gotarredona, and R. Serrano-Gotarredona. Compact low-power calibration mini-DACs for neural arrays with programmable weights.

IEEE Transactions on Neural Networks, 14(5):1207–1216, 2003. doi: 10.1109/TNN.2003.816370.

S-C. Liu and R. Douglas. Temporal coding in a silicon network of integrate-and-fire neurons. *IEEE Transactions on Neural Networks*, 15(5):1305–1314, Sept 2004. ISSN 1045-9227. doi: 10.1109/TNN.2004.832725.

S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas. *Event-based neuromorphic systems*. Wiley, 2015. doi: 10.1002/9781118927601.ch6.

Shih-Chii Liu, A. van Schaik, B.A. Minch, and T. Delbruck. Event-based 64-channel binaural silicon cochlea with q enhancement mechanisms. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2027–2030, May 2010. doi: 10.1109/ISCAS.2010.5537164.

Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/S0893-6080(97)00011-7. URL http://www.sciencedirect.com/science/article/pii/S0893608097000117.

Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4):593 – 616, 2004. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/j.jcss.2004.04.001. URL http://www.sciencedirect.com/science/article/pii/S0022000004000406.

Wolfgang Maass, Robert Legenstein, and Henry Markram. A new approach towards vision suggested by biologically realistic neural microcircuit models. In HeinrichH. Bülthoff, Christian Wallraven, Seong-Whan Lee, and TomasoA. Poggio, editors, *Biologically Motivated Computer Vision*, volume 2525 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg, 2002a. ISBN 978-3-540-00174-4. doi: 10.1007/3-540-36181-2_28. URL http://dx.doi.org/10.1007/3-540-36181-2_28.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–2560, November 2002b. ISSN 0899-7667. doi: 10.1162/089976602760407955. URL http://dx.doi.org/10.1162/089976602760407955.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. A model for real-time computation in generic neural microcircuits. In S. Thrun and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 213–220. MIT Press, Cambridge, MA, 2002c. URL http://books.nips.cc/papers/files/nips15/NS15.pdf.

Misha Mahowald. *An Analog VLSI System for Stereoscopic Vision*. Kluwer Academic Publishers, Norwell, MA, USA, 1994. ISBN 0792394445.

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

Henry Markram. The blue brain project. *Nat Rev Neurosci*, 7(2):153–160, 02 2006. URL http://dx.doi.org/10.1038/nrn1848.

Henry Markram, Yun Wang, and Misha Tsodyks. Differential signaling via the same axon of neocortical pyramidal neurons. *Proceedings of the National Academy of Sciences*, 95(9):5323–5328, 1998. URL http://www.pnas.org/content/95/9/5323.abstract.

Henry Markram, Karlheinz Meier, Thomas Lippert, Sten Grillner, Richard Frackowiak, Stanislas Dehaene, Alois Knoll, Haim Sompolinsky, Kris Verstreken, Javier DeFelipe, Seth Grant, Jean-Pierre Changeux, and Alois Saria. Introducing the human brain project. *Procedia Computer Science*, 7(0):39 – 42, 2011. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2011.12.015. URL http://www.sciencedirect.com/science/article/pii/S1877050911006806. Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11).

WarrenS. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. ISSN 0007-4985. doi: 10.1007/BF02478259. URL http://dx.doi.org/10.1007/BF02478259.

C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, Oct 1990. ISSN 0018-9219. doi: 10.1109/5.58356.

P. Merolla, J. Arthur, R. Alvarez, J-M. Bussat, and K. Boahen. A multicast tree router for multichip neuromorphic systems. *IEEE Trans. Circuits and Syst. I*, 61(3):820–833, March 2014a. doi: 10.1109/TCSI.2013.2284184.

Paul Merolla, Tristan Ursell, and John V. Arthur. The thermodynamic temperature of a rhythmic spiking network. *CoRR*, abs/1009.5473, 2010. URL http://arxiv.org/abs/1009.5473.

Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014b. doi: 10.1126/science.1254642. URL http://www.sciencemag.org/content/345/6197/668.abstract.

Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

Srinjoy Mitra, Stefano Fusi, and Giacomo Indiveri. Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI. *IEEE Trans. on Biomedical Circuits and Systems*, 3(1):32–42, 2009.

A. Mohamed, G. E. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *Trans. Audio, Speech and Lang. Proc.*, 20(1):14–22, January 2012. ISSN 1558-7916. doi: 10.1109/TASL.2011.2109382. URL http://dx.doi.org/10.1109/TASL.2011.2109382.

Don Monroe. Neuromorphic computing gets ready for the (really) big time. *Commun. ACM*, 57(6):13–15, June 2014. ISSN 0001-0782. doi: 10.1145/2601069. URL http://doi.acm.org/10.1145/2601069.

S.W. Moore, P.J. Fox, S.J.T. Marsh, A.T. Markettos, and A. Mujumdar. Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 133–140, May 2012. doi: 10.1109/FCCM.2012.32.

S. Moradi and G. Indiveri. An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE Trans. Biomed. Circuits Syst.*, 8(1):98–107, 2014. doi: 10.1109/TBCAS.2013.2255873.

D. Moratal. Principles of computational modelling in neuroscience (sterratt, d. et al.; 2011) [book reviews]. *Pulse, IEEE*, 3(4):82, July 2012. ISSN 2154-2287. doi: 10.1109/MPUL.2012.2196841.

A Morrison, C Mehring, T Geisel, A Aertsen, and M Diesmann. Advancing the boundaries of high-connectivity network simulation with distributed computing. *Neural Computation*, 17(8):1776–1801, Aug 2005. ISSN 0899-7667. doi: 10.1162/0899766054026648.

Abigail Morrison, Markus Diesmann, and Wulfram Gerstner. Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, 98(6):459–478, 06 2008. doi: 10.1007/s00422-008-0233-1. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2799003/.

A. Mundy, J. Knight, C.T. Stewart, and S. Furber. An efficient SpiNNaker implementation of the neural engineering framework. In *Accepted in 2015 International Joint Conference on Neural Networks (IJCNN)*, Jul 2015.

Jayram Moorkanikara Nageswaran, Nikil Dutt, Jeffrey L. Krichmar, Alex Nicolau, and Alex Veidenbaum. Efficient simulation of large-scale spiking neural networks using cuda graphics processors. In *Proceedings of the 2009 International Joint Conference on Neural Networks*, IJCNN'09, pages 3201–3208, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-3549-4. URL http://dl.acm.org/citation.cfm?id=1704555.1704736.

Javier Navaridas, Luis A. Plana, Jose Miguel-Alonso, Mikel Luján, and Steve B. Furber. Spinnaker: Impact of traffic locality, causality and burstiness on the performance of the interconnection network. In *Proceedings of the 7th ACM International Conference on Computing Frontiers*, CF '10, pages 11–20, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0044-5. doi: 10.1145/1787275.1787278. URL http://doi.acm.org/10.1145/1787275.1787278.

E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, 7(272), 2014. ISSN 1662-453X. doi: 10.3389/fnins.2013.00272. URL http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2013.00272/abstract.

Emre Neftci, Elisabetta Chicca, Giacomo Indiveri, and Rodney Douglas. A systematic method for configuring VLSI networks of spiking neurons. *Neural Computation*, 23 (10):2457–2497, 2011.

D. Neil and S.-C. Liu. Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12): 2621–2628, 2014. ISSN 1063-8210. doi: 10.1109/TVLSI.2013.2294916.

P. O'Connor, D. Neil, S-C. Liu, T. Delbruck, and M. Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, 7 (178), 2013. doi: 10.3389/fnins.2013.00178.

M. Oster, Y. Wang, R. Douglas, and S-C. Liu. Quantification of a spike-based winner-take-all VLSI network. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(10):3160–3169, Nov 2008. doi: 10.1109/TCSI.2008.923430.

Jongkil Park, T. Yu, C. Maier, S. Joshi, and G. Cauwenberghs. Live demonstration: Hierarchical address-event routing architecture for reconfigurable large scale neuromorphic systems. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 707–711, May 2012. doi: 10.1109/ISCAS.2012.6272133.

Cameron Patterson. *Managing a Real-Time Massively-Parallel Neural Architecture*. PhD thesis, School of Computer Science, University of Manchester, 2012.

Cameron Patterson, Jim Garside, Eustace Painkras, Steve Temple, Luis A. Plana, Javier Navaridas, Thomas Sharp, and Steve Furber. Scalable communications for a million-core neural processing architecture. *Journal of Parallel and Distributed Computing*, 72(11):1507 – 1520, 2012. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/j.jpdc.2012.01.016. URL `http://www.sciencedirect.com/science/article/pii/S0743731512000287`. Communication Architectures for Scalable Systems.

Hélène Paugam-Moisy, Régis Martinez, and Samy Bengio. Delay learning and polychronization for reservoir computing. *Neurocomputing*, 71(7–9):1143 – 1158, 2008. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2007.12.027. URL `http://www.sciencedirect.com/science/article/pii/S0925231208000507`. Progress in Modeling, Theory, and Application of Computational Intelligenc15th European Symposium on Artificial Neural Networks 200715th European Symposium on Artificial Neural Networks 2007.

Verena Pawlak, Jeffery R Wickens, Alfredo Kirkwood, and Jason N D Kerr. Timing is not everything: neuromodulation opens the STDP gate. *Frontiers in Synaptic Neuroscience*, 2, 2010.

M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: Principles and methods. *Proceedings of the IEEE*, 94(8):1487–1501, Aug 2006. ISSN 0018-9219. doi: 10.1109/JPROC.2006.879797.

M.J.M. Pelgrom, A.C.J. Duinmaijer, and A.P.G. Welbers. Matching properties of MOS transistors. *IEEE J. Solid-State Circuits*, 25(5):1212–1224, 1989.

J.A. Perez-Carrasco, Bo Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, Shouchun Chen, and B. Linares-Barranco. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(11):2706–2719, Nov 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2013.71.

Mihai A Petrovici, Johannes Bill, Ilja Bytschok, Johannes Schemmel, and Karlheinz Meier. Stochastic inference with deterministic spiking neurons. *arXiv preprint*, 1311.3211, 2013.

T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in Neuroscience*, 7:11, 2013. doi: 10.3389/fnins.2013.00011.

Thomas Pfeil, Tobias C Potjans, Sven Schrader, Wiebke Potjans, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier. Is a 4-bit synaptic weight resolution enough?–constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Frontiers in neuroscience*, 6, 2012. doi: 10.3389/fnins.2012.00090.

P-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello. Neuflow: Dataflow vision processing system-on-a-chip. In *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1044–1047, 2011.

H. E Plesser, J. M Eppler, A. Morrison, M. Diesmann, and M.-O. M.O. Gewaltig. Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In *Proc. 13th Int'l Euro-Par Conf. on Parallel Processing*

(Euro-Par 2007), pages 672–681. Springer, 2007. URL http://www.springerlink.com/index/RW27162113P7L685.pdf.

W Potjans, M Diesmann, and A Morrison. An Imperfect Dopaminergic Error Signal Can Drive Temporal-Difference Learning. *PLoS Computational Biology*, 7(5): 20, 2011. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3093351\&tool=pmcentrez\&rendertype=abstract.

Robert Preissl, Theodore M. Wong, Pallab Datta, Myron Flickner, Raghavendra Singh, Steven K. Esser, William P. Risk, Horst D. Simon, and Dharmendra S. Modha. Compass: A scalable simulator for an architecture for cognitive computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 54:1–54:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press. ISBN 978-1-4673-0804-5. URL http://dl.acm.org/citation.cfm?id=2388996.2389070.

Dimitri Probst, Wolfgang Maass, Henry Markram, and Marc-Oliver Gewaltig. Liquid computing in a simplified model of cortical layer iv: Learning to balance a ball. In *Proceedings of the 22Nd International Conference on Artificial Neural Networks and Machine Learning - Volume Part I*, ICANN'12, pages 209–216, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33268-5. doi: 10.1007/978-3-642-33269-2_27. URL http://dx.doi.org/10.1007/978-3-642-33269-2_27.

D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat. Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Trans on Nanotechnology*, 12(3):288–295, 2013.

Alexander Rast, Francesco Galluppi, Sergio Davies, Luis Plana, Cameron Patterson, Thomas Sharp, David Lester, and Steve Furber. Concurrent heterogeneous neural model simulation on real-time neuromimetic hardware. *Neural Networks*, 24 (9):961 – 978, 2011. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2011.06.014. URL http://www.sciencedirect.com/science/article/pii/S0893608011001742. Multi-Scale, Multi-Modal Neural Modeling and Simulation.

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

J. Sawada and D.S. Modha. SyNAPSE: Scalable Energy-Efficient Neurosynaptic

Computing. In *Application of Concurrency to System Design (ACSD), 2013 13th International Conference on*, pages xiv–xv, July 2013. doi: 10.1109/ACSD.2013.36.

J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 431–438, June 2008. doi: 10.1109/IJCNN.2008.4633828.

J. Schemmel, D. Brüderle, A. Grübl, 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*, pages 1947–1950, May 2010. doi: 10.1109/ISCAS.2010.5536970.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. URL `http://www.sciencedirect.com/science/article/pii/S0893608014002135`.

Terrence Sejnowski. The computational self. *Annals of the New York Academy of Sciences*, 1001(1):262–271, 2003. ISSN 1749-6632. doi: 10.1196/annals.1279.015. URL `http://dx.doi.org/10.1196/annals.1279.015`.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimenez, C. Serrano-Gotarredona, J.A. Perez-Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jimenez-Moreno, and A. Civit-Ballcels. On real-time AER 2-D convolutions hardware for neuromorphic spike-based cortical processing. *IEEE Transactions on Neural Networks*, 19(7):1196–1219, July 2008. ISSN 1045-9227. doi: 10.1109/TNN.2008.2000163.

T. Serrano-Gotarredona and B. Linares-Barranco. Systematic width-and-length dependent CMOS transistor mismatch characterization and simulation. *Analog Integrated Circuits and Signal Processing*, 21:271–296, 1999.

T. Sharp and S. Furber. Correctness and performance of the spinnaker architecture. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8, Aug 2013. doi: 10.1109/IJCNN.2013.6706988.

Thomas Sharp, LuisA. Plana, Francesco Galluppi, and Steve Furber. Event-Driven Simulation of Arbitrary Spiking Neural Networks on SpiNNaker. In Bao-Liang Lu, Liqing Zhang, and James Kwok, editors, *Neural Information Processing*, volume 7064 of *Lecture Notes in Computer Science*, pages 424–430. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24964-8. doi: 10.1007/978-3-642-24965-5_48.

Thomas Sharp, Francesco Galluppi, Alexander Rast, and Steve Furber. Power-efficient simulation of detailed cortical microcircuits on spinnaker. *Journal of Neuroscience Methods*, 210(1):110 – 118, 2012. ISSN 0165-0270. doi: 10.1016/j.jneumeth.2012.03.001. URL http://www.sciencedirect.com/science/article/pii/S0165027012000866. Special Issue on Computational Neuroscience.

Thomas Sharp, Rasmus Petersen, and Steve Furber. Real-time million-synapse simulation of rat barrel cortex. *Frontiers in Neuroscience*, 8:131, 2014. doi: 10.3389/fnins.2014.00131. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4038760/.

A. J. F. Siegert. On the first passage time probability problem. *Physical Review*, 81(4):617, 1951.

Rae Silver, Kwabena Boahen, Sten Grillner, Nancy Kopell, and Kathie L Olsen. Neurotech for neuroscience: unifying concepts, organizing principles, and emerging tools. *Journal of Neuroscience*, 27(44):11807–11819, 2007. URL http://www.ncbi.nlm.nih.gov/pubmed/17978017.

P J Sjöström, G G Turrigiano, and S B Nelson. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron*, 32:1149–1164, 2001.

S Song and L F Abbott. Cortical development and remapping through spike timing-dependent plasticity. *Neuron*, 32(2):339–350, October 2001.

S Song, K D Miller, and L F Abbott. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, September 2000. doi: 10.1038/78829. URL http://dx.doi.org/10.1038/78829.

Terrence C Stewart, Bryan Tripp, and Chris Eliasmith. Python scripting in the nengo simulator. *Frontiers in Neuroinformatics*, 3(7), 2009. ISSN 1662-5196. doi: 10.3389/neuro.11.007.2009. URL http://www.frontiersin.org/neuroinformatics/10.3389/neuro.11.007.2009/abstract.

E. Stromatias, F. Galluppi, C. Patterson, and S. Furber. Power analysis of large-scale, real-time neural networks on spiNNaker. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Aug 2013. doi: 10.1109/IJCNN.2013.6706927.

E. Stromatias, C. Patterson, and S. Furber. Optimising the overall power usage on the spinnaker neuromimetic platform. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 4280–4287, July 2014. doi: 10.1109/IJCNN.2014.6889837.

E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S-C. Liu, and S. Furber. Scalable energy-efficient, low-latency implementations of spiking deep belief networks on SpiNNaker. In *Accepted in 2015 International Joint Conference on Neural Networks (IJCNN)*, Jul 2015a.

E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S-C. Liu, and S. Furber. Live demonstration: Handwritten digit recognition using Spiking Deep Belief Networks on SpiNNaker. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015b.

Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in Neuroscience*, 9(222), 2015c. ISSN 1662-453X. doi: 10.3389/fnins.2015.00222. URL `http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2015.00222/abstract`.

H. A. Swadlow. Physiological properties of individual cerebral axons studied in vivo for as long as one year. *Journal of Neurophysiology*, 54(5):1346–1362, 1985. ISSN 0022-3077.

Olivier Temam. A defect-tolerant accelerator for emerging high-performance applications. In *Proceedings of the 39th International Symposium on Computer Architecture*, ISCA '12, pages 356–367, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4503-1642-2. URL `http://dl.acm.org/citation.cfm?id=2337159.2337200`.

Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 06 1996. URL `http://dx.doi.org/10.1038/381520a0`.

Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14(6–7):715 – 725, 2001. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/S0893-6080(01)00083-1. URL http://www.sciencedirect.com/science/article/pii/S0893608001000831.

T.P Trappenberg. *Continuous attractor neural networks*. Recent Developments in Biologically Inspired Computing', Leandro Nunes de Castro & Fernando J. Von Zuben (eds.), IDEA Group Publishing, 2005. ISBN 1-59140-313-8.

K. Underwood. FPGAs vs. CPUs: Trends in peak floating-point performance. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, 2004.

D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521 – 528, 2005. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/j.ipl.2005.05.019. URL http://www.sciencedirect.com/science/article/pii/S0020019005001523. Applications of Spiking Neural Networks.

D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391 – 403, 2007. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2007.04.003. URL http://www.sciencedirect.com/science/article/pii/S089360800700038X. Echo State Networks and Liquid State Machines.

R.J. Vogelstein, U. Mallik, J.T. Vogelstein, and G. Cauwenberghs. Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses. *IEEE Transactions on Neural Networks*, 18(1):253–265, 2007.

C Vreeswijk, L F Abbott, and G Bard Ermentrout. When inhibition not excitation synchronizes neural firing. *J Comp Neurosci*, 1(4):313–321, 1994.

Y-X. Wang and S-C. Liu. Programmable synaptic weights for an aVLSI network of spiking neurons. In *Proc. IEEE Int. Symp. Circuits and Syst.*, 2006. doi: 10.1109/ISCAS.2006.1693637.

Y-X. Wang and S-C. Liu. A two-dimensional configurable active silicon dendritic neuron array. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(9):2159–2171, Sep 2011. doi: 10.1109/TCSI.2011.2112570.

Y-X. Wang and S-C. Liu. Active processing of spatio-temporal input patterns in silicon dendrites. *IEEE Transactions on Biomedical Circuits and Systems*, 7(3):307–318, June 2013. ISSN 1932-4545. doi: 10.1109/TBCAS.2012.2199487.

P. Werbos. Beyond regression: new tools for prediction and analysis in the behavioral sciences. 1974.

Theodore M. Wong, Robert Preissl, Pallab Datta, Myron Flickner, Raghavendra Singh, Steven K. Esser, Emmett McQuinn, Rathinakumar Appuswamy, William P. Risk, Horst D. Simon, and Dharmendra S. Modha. Ten to power 14. Technical Report RJ10502, IBM, April 2013. URL `http://domino.research.ibm.com/library/cyberdig.nsf/papers/19B9020D53E753DB85257AB7005FFA18`.

Jian Wu and Steve Furber. A multicast routing scheme for a universal spiking neural network architecture. *Comput. J.*, 53(3):280–288, March 2010. ISSN 0010-4620. doi: 10.1093/comjnl/bxp024. URL `http://dx.doi.org/10.1093/comjnl/bxp024`.

Theodore Yu, Jongkil Park, Siddharth Joshi, Christoph Maier, and Gert Cauwenberghs. 65k-neuron integrate-and-fire array transceiver with address-event reconfigurable synaptic routing. In *Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE*, pages 21–24. IEEE, 2012.

D. Yudanov and L. Reznik. Scalable multi-precision simulation of spiking neural networks on gpu with opencl. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, June 2012. doi: 10.1109/IJCNN.2012.6252440.

Appendix A

Power Characterisation - Supplementary Materials

Extra figures from Chapter 4 recorded using a Tektronix TDS 3034B oscilloscope.

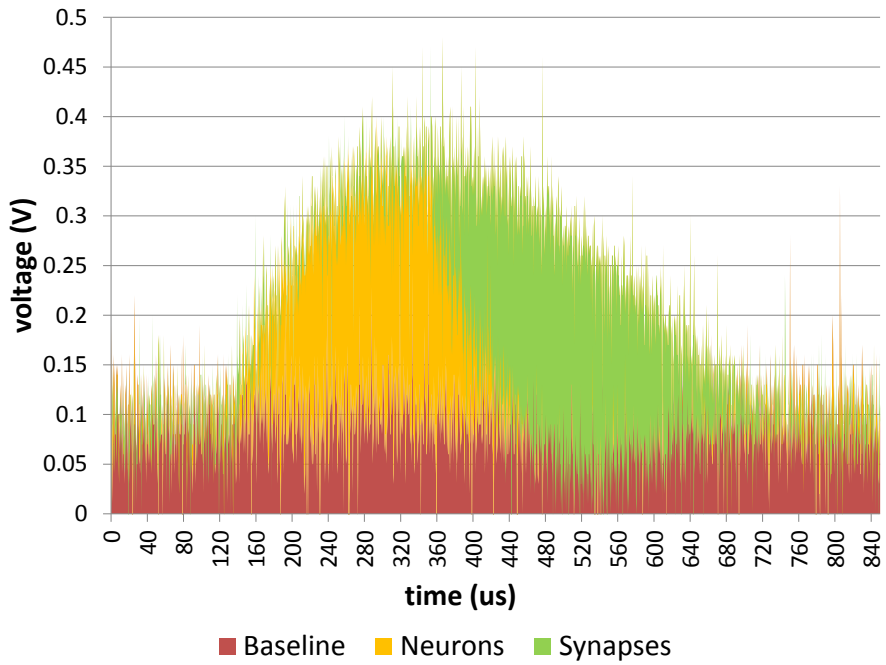


Figure A.1: Instantaneous voltage of the 1.2V(A) regulator, for 1 ms of simulation for the self-connected benchmark network with 326 LIF neurons per population (core).

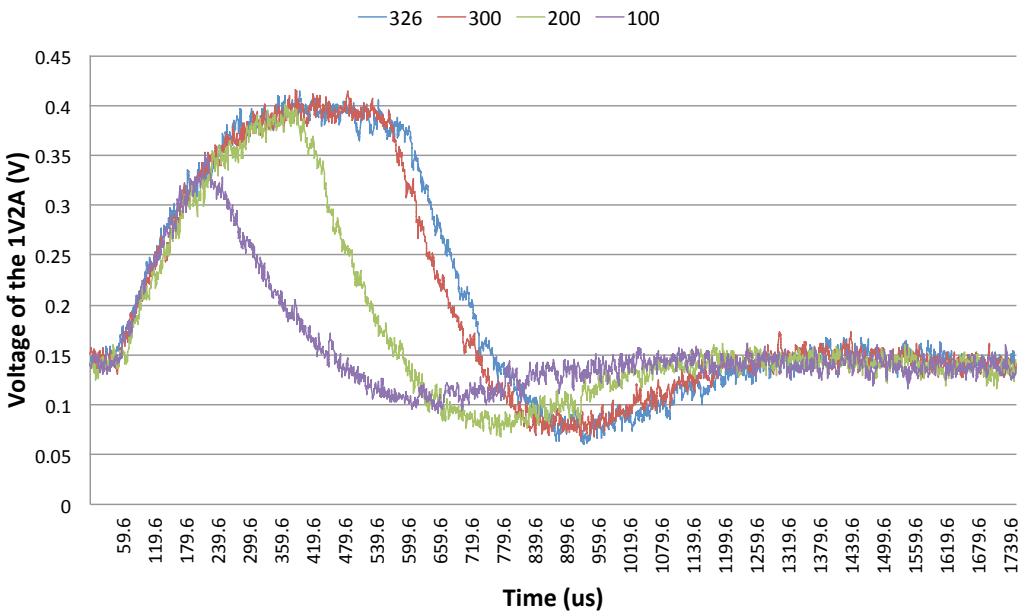


Figure A.2: Instantaneous voltage of the 1.2V(A) regulator, for 1 ms of simulation for the self-connected benchmark network, with different number of LIF neurons per core. The current injected to neurons ($I_{injected}$) was set to zero. Data are filtered with a low-pass filter.

Appendix B

NEST vs SpiNNaker

This section shows that SpiNNaker outperforms the reference multi-threaded simulator NEST [Gewaltig and Diesmann, 2007; Morrison et al., 2005] for the benchmark neural networks presented in Section 4.1.2. NEST (2.2.2) compiled with OpenMP [Dagum and Menon, 1998] ran on a high-performance multicore machine that consists of 384 GB of memory and 2 $\times$ Intel Xeon CPU E5-2690 processors, each with 8 cores (16 threads), clocked at 2.9 GHz. The SpiNN-4 board consists of 48 SpiNNaker chips, 864 ARM9 cores of which 768 were used during the simulations. The cores were clocked at 200 MHz, the routers at 100 MHz and the memory clocks at 133 MHz. The Python cProfile module was used to record the execution profile of each simulation. The same PyNN script was used for both platforms. Table B.1 summarises the two hardware platforms, while Table B.2 shows the software components used in the simulations.

Results show that for the locally-connected network, depicted in Figure 4.1, the single threaded NEST rans 48 $\times$ slower than real-time. NEST's execution time improves with the addition of extra threads; with 16 threads NEST rans 22 $\times$ slower than real-time, however beyond that point, using more threads does not seem to affect the execution time. A similar behaviour can be observed for the randomly-connected network, presented in Figure 4.2. NEST running on a single thread executes 45 $\times$ slower than real-time. Increasing the number of threads improves the execution time up to a point, which for this particular case is 16 $\times$ slower than real-time for 12 threads, while increasing the number of threads does not improve the performance of NEST. This behaviour is due to the fact that the two processors share a memory bus [Intel, 2014]. Results indicate that NEST is not capable of outperforming SpiNNaker with any number of processors. Moreover, NEST draws approximately seven times the power and four times the computational resources of SpiNNaker. A single Intel Xeon CPU

E5-2690 processor draws 135 W [Intel, 2014] and can deliver 371 gigaoperations per second (GOPS) [Intel Xeon Processor E5-2600 Series, 2014], while 768 SpiNNaker cores draw 36.37 W and can deliver approximately 153.6 (GOPS).

Table B.1: Hardware platform summary

Hardware	**Processing**	**Memory**	**OS**
Multicore	Intel Xeon CPU E5-2690 32 threads on 16 cores (2 chips) 2.9GHz clock	20MB Cache 384GB	Ubuntu 14.04.1
SpiNNaker	48 SpiNNaker chip board 784 cores on 48 chips 200MHz	32KB for instructions 64KB for data per core 128MB data per chip	SARK (v1.09)

Table B.2: Software component summary.

Software	**Version**	**Notes**
Python	2.7	with NumPy 1.8.2 and Scipy 0.13.3
NEST	2.2.2	with OpenMP 3.1
PyNN	0.75	

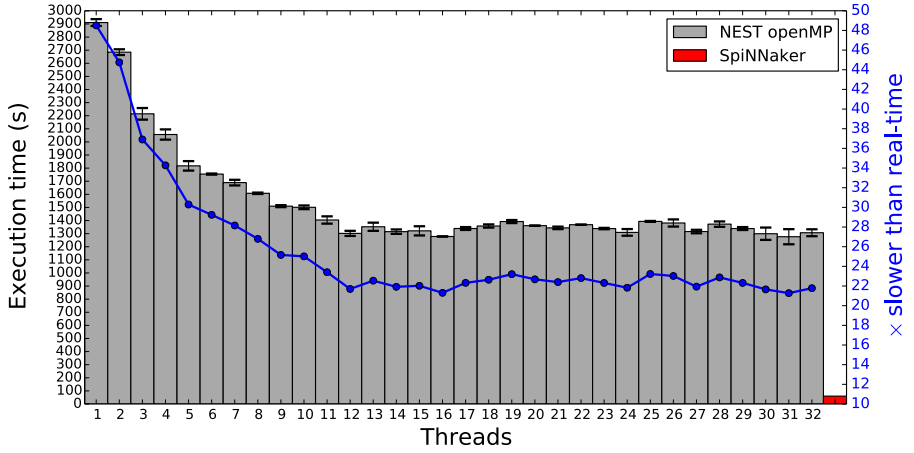


Figure B.1: Execution time of the 60-second locally-connected benchmark network with 326 neurons per population.

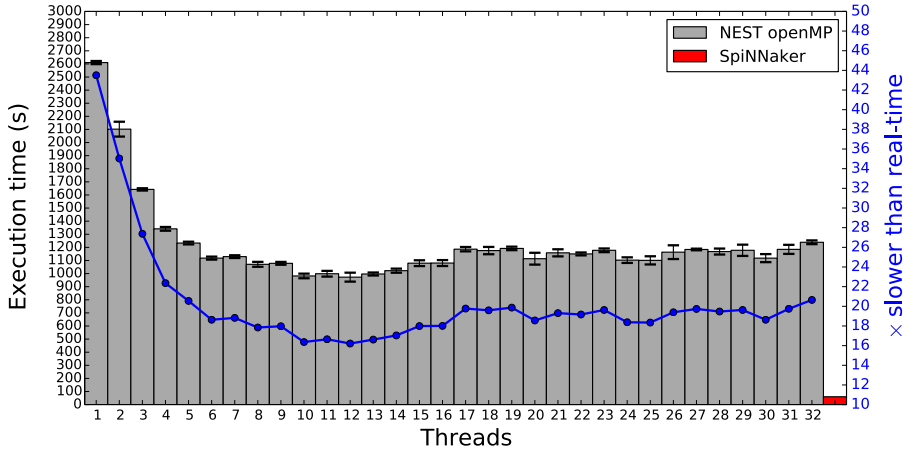


Figure B.2: Execution time of the 60-second randomly-connected benchmark network with 150 neurons per population.

Appendix C

Spiking Deep Belief Networks - Supplementary Materials

C.1 Training Scripts

This section presents the Matlab scripts used to train the two spike-based DBNs of Chapter 5. Section C.1.1 shows the identical training script used to train the DBN with the 2 hidden layers as published by O'Connor et al. [2013] and Neil and Liu [2014]. Section C.1.2 presents the script used to train the novel DBN with the 7 hidden layers. The remaining Matlab files can be found at `https://github.com/dannyneil/edbn/`.

C.1.1 Training Script for the DBN With the 2 Hidden Layers

```
%% Load paths
addpath(genpath('.'));

%% Load data
load mnist_uint8;

train_x = double(train_x) / 255 * 0.2;
test_x  = double(test_x)  / 255 * 0.2;
train_y = double(train_y) * 0.2;
test_y  = double(test_y)  * 0.2;

%% Train network
```

```
rand('seed', 42);
clear edbn opts;
edbn.sizes = [784 500 500 10];
opts.numepochs = 2;
opts.alpha = 0.005;
[edbn, opts] = edbnsetup(edbn, opts);

opts.momentum = 0.0; opts.numepochs =  2;
edbn = edbntrain(edbn, train_x, opts);
edbn = edbntoptrain(edbn, train_x, opts, train_y);

 opts.momentum = 0.8; opts.numepochs = 60;
edbn = edbntrain(edbn, train_x, opts);

edbn = edbntrain(edbn, train_x, opts);
edbn = edbntoptrain(edbn, train_x, opts, train_y);

% Show results
figure;
visualize(edbn.erbm{1}.W');   % Visualize the RBM weights
er = edbntest (edbn, train_x, train_y);
fprintf('Scored: %2.2f\n', (1-er)*100);
filename = sprintf('good_mnist_%2.2f-%s.mat',(1-er)*100, date());
edbnclean(edbn);
save(filename,'edbn');

opts.momentum = 0.8;
opts.numepochs = 80;
edbn = edbntoptrain(edbn, train_x, opts, train_y);

% Show results
figure;
visualize(edbn.erbm{1}.W');   % Visualize the RBM weights
er = edbntest (edbn, train_x, train_y);
fprintf('Scored: %2.2f\n', (1-er)*100);
```

```
filename = sprintf('good_mnist_%2.2f-%s.mat',(1-er)*100, date());
edbnclean(edbn);
save(filename,'edbn');

%% Show the EDBN in action
spike_list = live_edbn(edbn, test_x(1, :), opts);
output_idxs = (spike_list.layers == numel(edbn.sizes));

figure(2); clf;
hist(spike_list.addrs(output_idxs) - 1, 0:edbn.sizes(end));

%% Export to xml to load into JSpikeStack
edbntoxml(edbn, opts, 'mnist_edbn');
```

C.1.2 Training Script for the DBN With the 7 Hidden Layers

```
%% Load paths
addpath(genpath('.'));

%% Load data
load mnist_uint8;

% Convert data and rescale between 0 and 0.2
train_x = double(train_x) / 255 * 0.2;
test_x  = double(test_x)  / 255 * 0.2;
train_y = double(train_y) * 0.2;
test_y  = double(test_y)  * 0.2;

%% Train network
% Setup
seed=33000;
rng(seed,'twister')

clear edbn opts;
edbn.sizes = [784 500 500 500 500 500 500 500 10];
opts.numepochs = 10;
```

```
[edbn, opts] = edbnsetup(edbn, opts);

% Train
fprintf('Beginning training.\n');
edbn = edbntrain(edbn, train_x, opts);
% Use supervised training on the top layer
edbn = edbntoptrain(edbn, train_x, opts, train_y);

edbn = edbntrain(edbn, train_x, opts);
% Use supervised training on the top layer
edbn = edbntoptrain(edbn, train_x, opts, train_y);

edbn = edbntrain(edbn, train_x, opts);
% Use supervised training on the top layer
edbn = edbntoptrain(edbn, train_x, opts, train_y);


% Show results
figure;
visualize(edbn.erbm{1}.W');   % Visualize the RBM weights

[er,bad] = edbntest (edbn, test_x, test_y);
fprintf('Scored: %2.2f\n', (1-er)*100);

%% Show the EDBN in action
spike_list = live_edbn(edbn, test_x(1, :), opts);
output_idxs = (spike_list.layers == numel(edbn.sizes));

figure(2); clf;
hist(spike_list.addrs(output_idxs) - 1, 0:edbn.sizes(end));
title('Label Layer Classification Spikes');
%% Export to xml
edbntoxml(edbn, opts, 'mnist_edbn');
```

C.2 Additional Figures for the DBN with 2 hidden layers

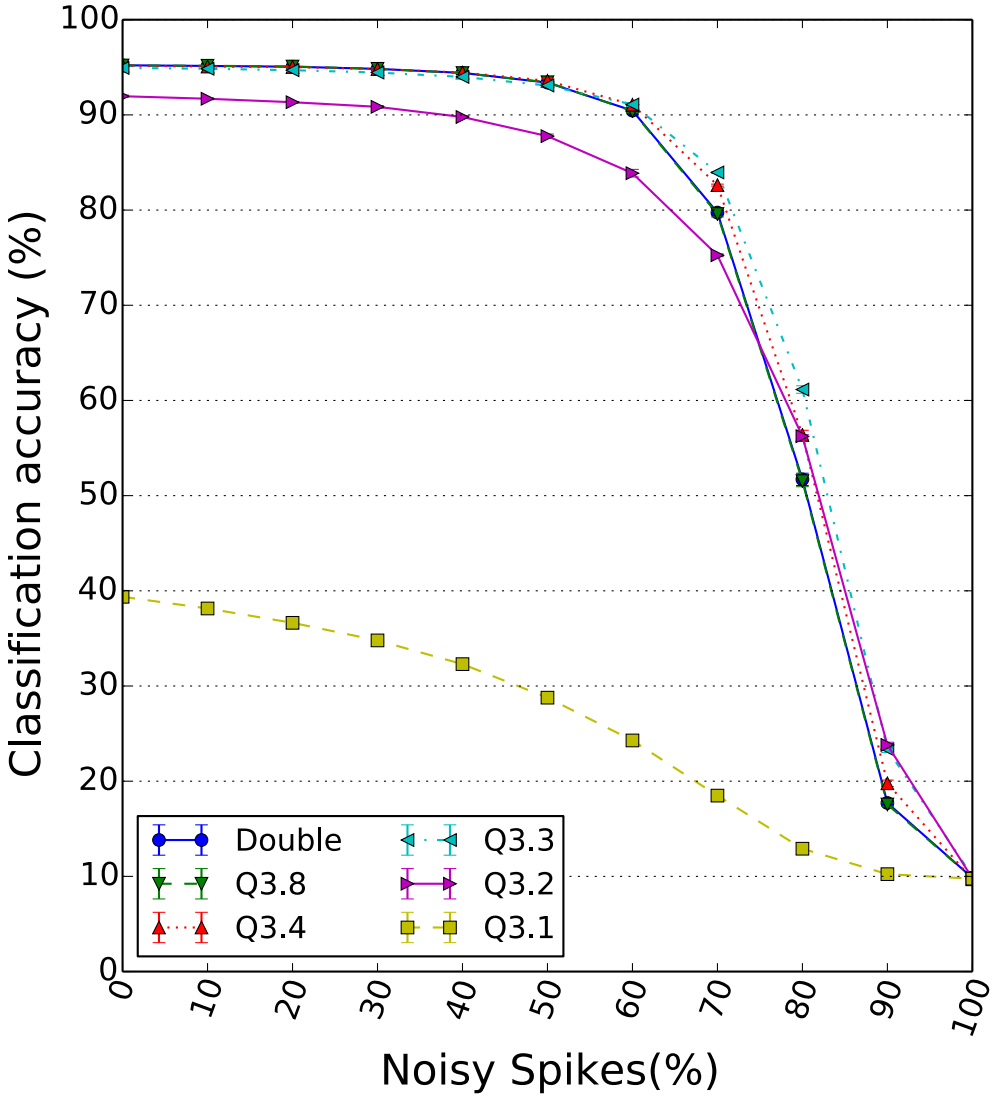


Figure C.1: CA for the DBN with the 2 hidden layers as a function of input noise and bit precision of synaptic weights for 100 spikes per second and a stimulus duration of 15 seconds. The performance is almost identical to the 1500 Hz case for a stimulus duration of 1 second.

C.3 Additional Figures for the DBN with 7 hidden layers

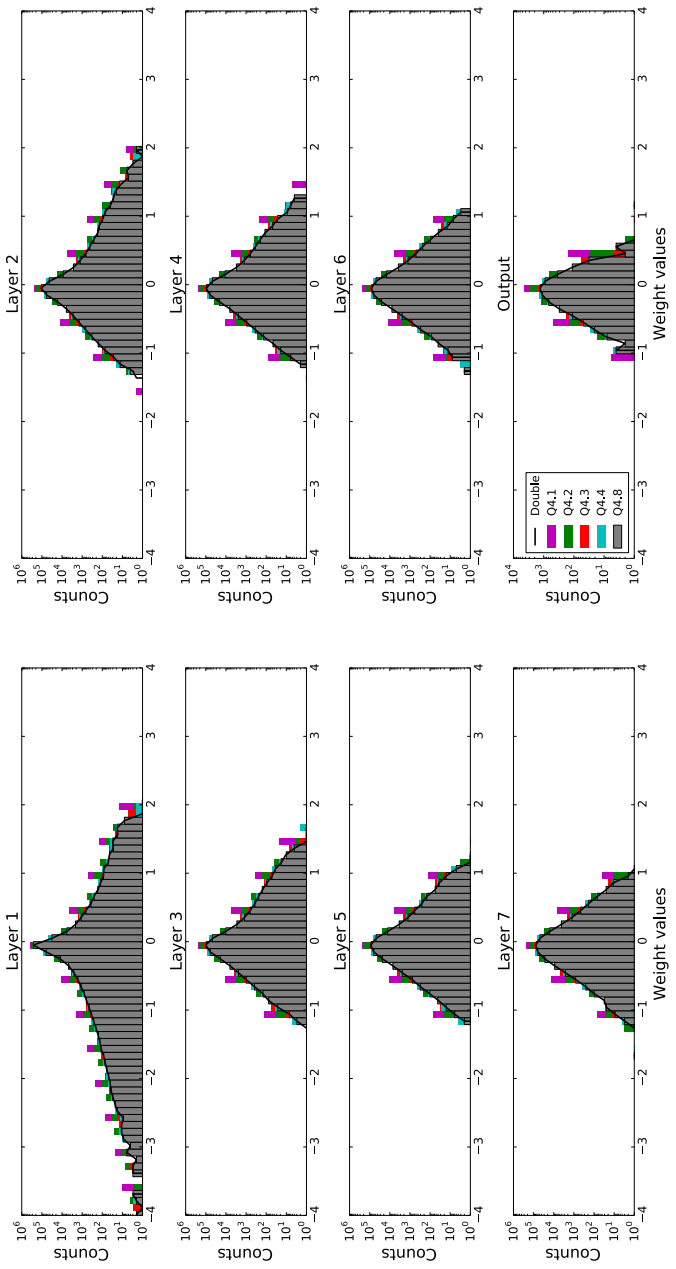


Figure C.2: Weight distribution of the DBN with the 7 hidden layers for different fixed-point representations.

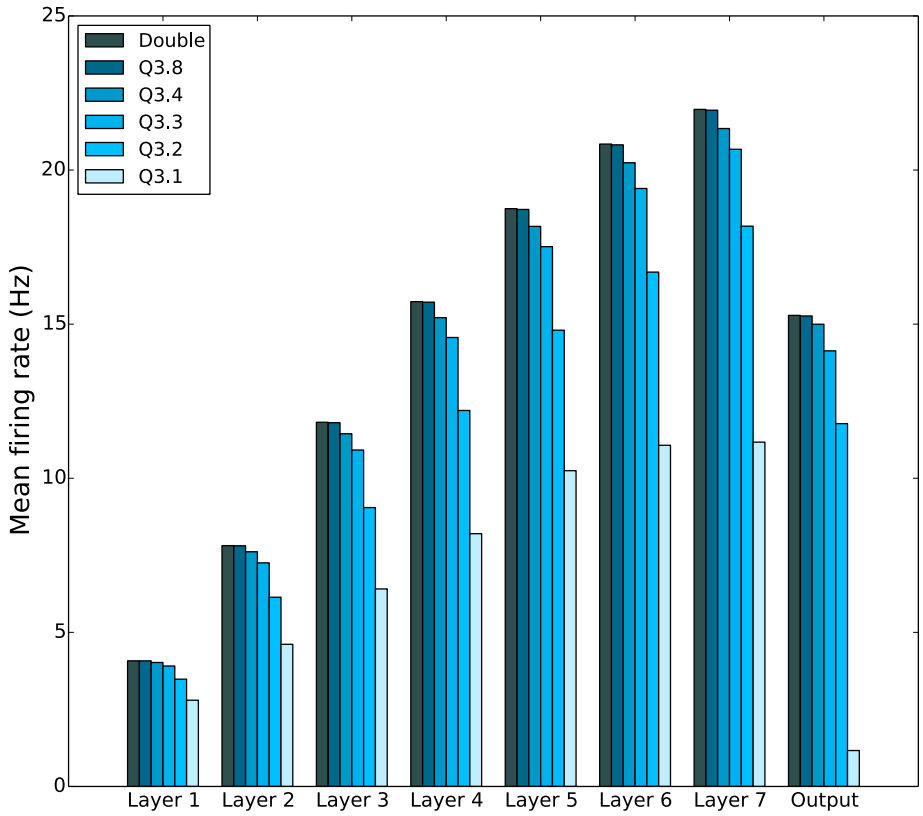


Figure C.3: Mean firing rate of the DBN with the 7 hidden layers for each layer of the network, for weights with different bit precisions, using an input rate of 1500Hz. Lower precision levels, which lead to more weights at zero, cause lower firing rates within the network.

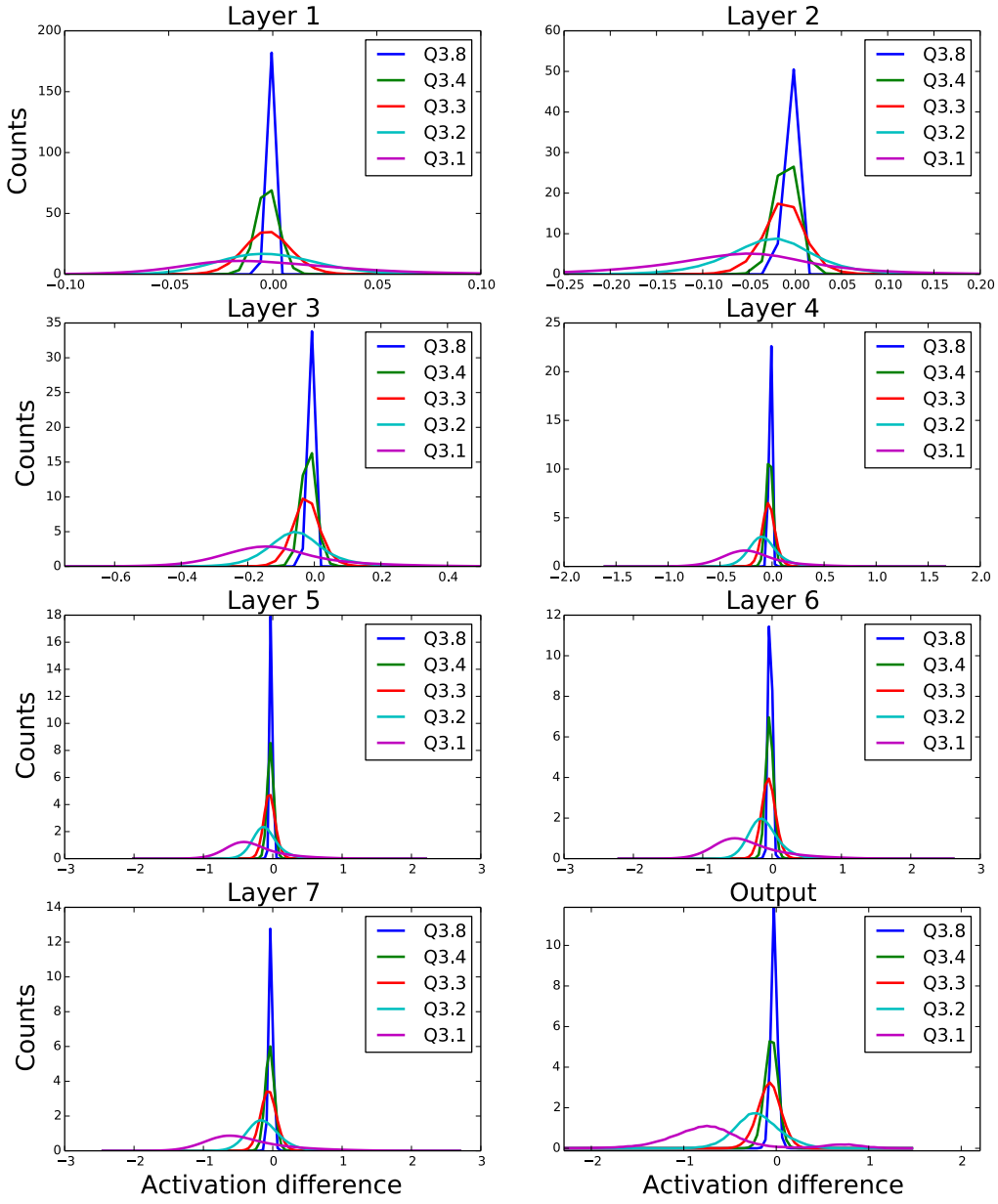


Figure C.4: Distribution of the mean difference, of the DBN with the 7 hidden layers, between the activation of a neuron with double precision weights and neurons using weights with different bit precision levels. Shown are distributions over all test samples. The difference, although peaked near zero, increases for higher layers, and shows a trend towards reduced activations.

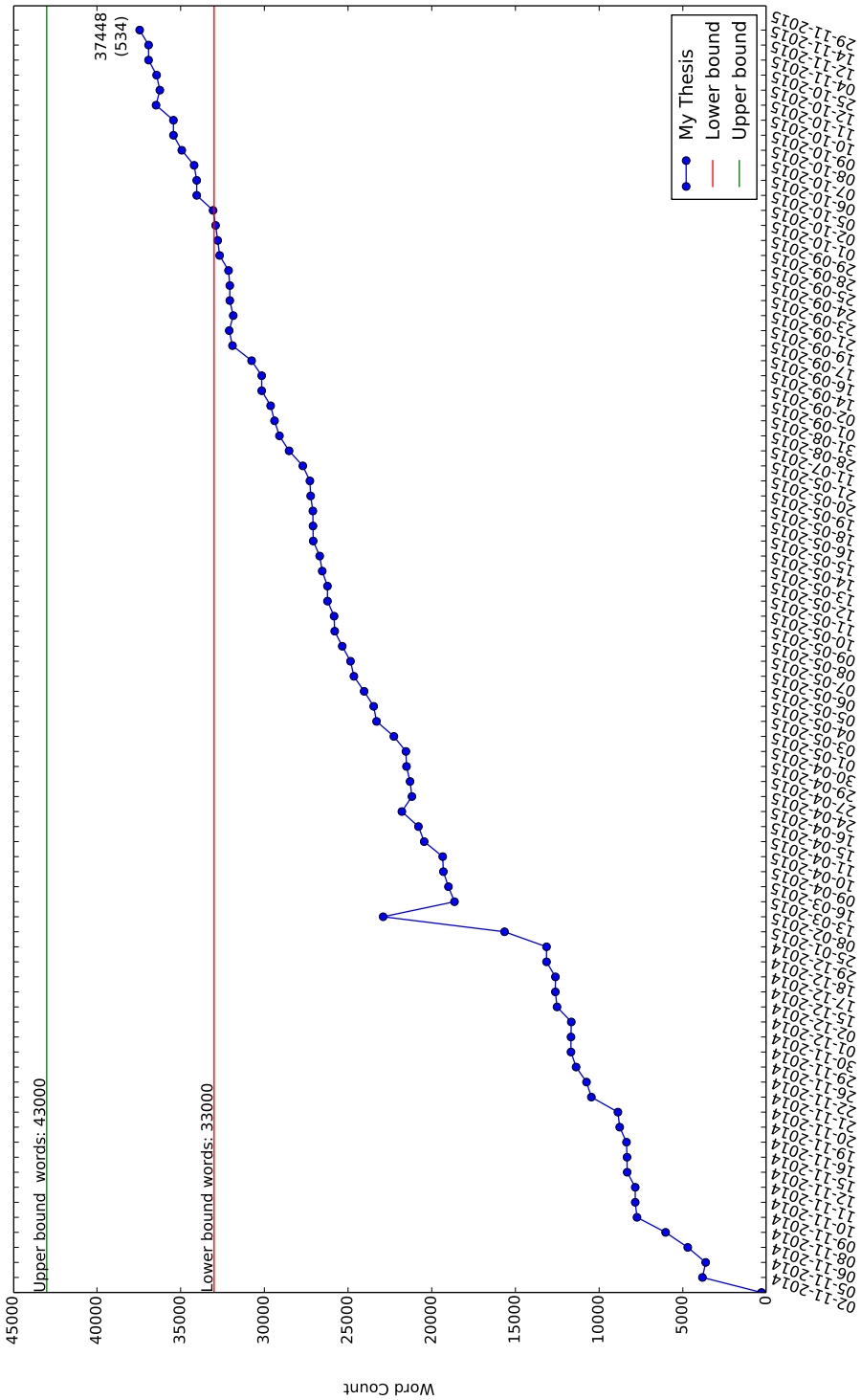


Figure C.5: Thesis progression.