STOCHASTIC PROCESSES FOR NEUROMORPHIC HARDWARE

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

2020

By
Gabriel Andres Fonseca Guerra
Department of Computer Science

Contents

List of Tables

List of Figures

Abstract

STOCHASTIC PROCESSES FOR NEUROMORPHIC HARDWARE
Gabriel Andres Fonseca Guerra
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2020

Neuromorphic technology is evolving rapidly, but it still faces two critical problems. Firstly, few compelling applications exist that demonstrate the superiority of neuromorphic technology over classical computing, limiting its widespread adoption and commercialisation. Some insightful applications include keyword spotting [BCHE19], real-time modelling of microcortical circuits [RPR$^+$20] the implementation of nearest-neighbor searches [FOF$^+$20] and LASSO optimisation via the spiking locally competityive algorithm [DSL$^+$18]. Secondly, the use of neuromorphic technology by neuroscientists is scarce, with physicists, mathematicians, engineers and computer scientists as the principal user communities. Discrepancies remain between the variables of interest in the laboratory to experimental neuroscientists and the parameterisations realisable on neuromorphic hardware, making the models of the latter too abstract or simplified. For example, while experimental data is acquired in the form of ion-channel conductances from patchlamp experiments, local field potentials, effects of parmacological blockers and neurotransmitter on neurons, and intra-cellular and extra-cellular ion concentrations, the neuromorphic hardware is configured in terms of synapse level connectivity (point-to-point adjacency matrices), membrane and postsynaptic potential time constants, inter spike intervals and firing probabilities. We contribute to addressing both issues by implementing stochastic processes arising in neuronal dynamics, developing applications for neuromorphic hardware of both biological and technological interest.

On the application level, we harness recent theoretical developments and results from conventional hardware on the computational power of stochastic neuronal dynamics for problem-solving. We do so by replicating and improving on the solution of constraint satisfaction problems (CSPs) with stochastic networks of spiking neurons. For this, we have used both the SpiNNaker and the Loihi neuromorphic chips, harnessing the advantages of each one. Our results demonstrate the usability of neuromorphic technology to solve hard problems with industrial application for which conventional machine learning faces challenges. The performance of our CSP solver is comparable to that of the state of the art solutions, and is a basic module for implementing solution strategies of increasing sophistication as well as for gaining insights into how living beings solve CSP problems in the real world. To bridge the gap with experimental neuroscience, we demonstrate the implementation on SpiNNaker of models of the intrinsic currents generated by voltage-gated ion channels, as well as of realistic postsynaptic potentials. Both of these arise in the neuronal membrane from complex ion-channel dynamics which are stochastic by their very nature. Our work paves the way to integrate neuromorphic technology with the worlds of neurophysiology and neurogenetics, allowing a direct relation with processes of interest in neuropharmacology, such as protein-drug interaction, as well as in whole-cell recordings of phenomena such as homeostasis and intrinsic plasticity. Hence these results at the cellular level open the way for the use of neuromorphics in medical applications and scientific enterprise in neuroscience.

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 thank the continuous support and guidance from my supervisor Professor Steve Furber, who gave me the opportunity to travel and meet with great scientists all along my PhD, transcending the University of Manchester. He provided the flexibility for me to move horizontally across research fields which created a trade-off between expertise in a narrow field and critical thinking, transferable skill and knowledge of what is available there outside (in biology, medicine, electrical engineering, philosophy, chemistry...) to tackle the hardest problems of nature, which are too big for the methods of a single discipline. Sometimes, to even be aware of the existence of some open problems, one needs to get out of the disciplinary box, an action that provides the greatest delight.

This thesis would not be possible without the scientific and technical guidance from PhD Andreas Wild, my internship supervisor from Intel Labs, whose continuous effort, patience and inspiration became essential for the development of chapter 4 of this thesis: the development of a neuromorphic solver for constraint satisfaction problems with on-chip validation. We continued working together beyond the internship itself for almost a year now. During this time Andreas has taught me how to be certain and precise in the uncertain scientific enterprise of combining computer hardware and neuroscience. Such an interaction could only be possible thanks to Mike Davies our manager and director of the Neuromorphic Computing Lab at Intel Corporation. A never-stop researcher and hardware architect whose approach to the field is becoming a game-changer, pushing neuromorphics beyond its limits so that these finally reach society. Mike gave me the opportunity to spend more than three months in his lab, working as part of the Loihi software team, an unforgettable experience. I am grateful for the hospitality and support from the Loihi folks in Hillsboro and Santa Clara. I always felt like a member of the team and learnt so much from each of you.

I am sincerely grateful to the inspirational and challenging conversations, guidance and feedback from professors, Wolfgang Maass, Johan F. Storm, Karl Friston, Eva

Navarro, André Grüning, Jacob Sherson, as well as from Professor Karlheinz Meier who although no longer with us, continues living through his enthusiasm and inspiration to keeping neuromorphic hardware alive.

Although our conversations were shorter and less recurrent, the internet has made the impact equally valuable and essential from professors Yves Frgnac, Alain Destexhe Christoph Schwarzer, John Huguenard, Kathrin Otrel-Cass, as well as from the organisers of the CapoCaccia workshop, professors Giacomo Indivery, Rodney Douglas, Yulia Sandamirskaya, Kevan Martin, Tobi Delbruck, Shih-Chii Liu, and Terrence Sejnowski

To the everlasting love of my Family. My mom has always been the greatest example of curiosity and creativity and my dad of passion and discipline, but foremost, they have always given unconditional love, this thesis is dedicated to both of you. My dear siblings Dani, Caro, Ingrid, Lili, Lolo and Javi, you have always been there with love and motivation in the hardest moments.

This adventure really began with my wife back in sunny Brazil, only she knows the obstacles and challenges we had to overcome to reach this point, thanks Jaque for wildly pursuing our dreams together.

Thanks to my friends in Manchester, Gengting, Raul, Robert, Gary, Guillermo, Merve, Seckin, Mantas and Oliver and those from the Human Brain Project, Angelina Lesnikova, Eric Muller, Sacha van Albada, Eduarda Susin. Thanks to my research colleagues in the SpiNNaker project, Basab, Simon, Michael, Andrew, Luis and colleagues from Colombia, Fausto Lagos, Prof Jairo Arbey, and especially to Alain Riveros for his friendship and the horizons we have been drawing for the future.

Just as importantly, my friends Kirsten and Thyge made the last year of this PhD a whole new experience, tusind tak.

This work was only possible thanks to the generous support from the Brazilian National Council for Scientific and Technological Development (CNPq).

Chapter 1

Introduction

1.1 Motivation

Through the years, automated machines have improved manufacturing and other processes in all areas of human life. Initially, such devices were programmed to do repetitive tasks, more like a powerful and sophisticated calculator. However, nowadays we count with computer applications that can perform more cognitive-like tasks. These learn and are able to deal 'reliably' with problems they never saw before. Some of the most innovative and impressive technologies offered by industries like IBM [Hig12, CAW16], Google [SHM+16, SSS+17], Amazon [Jos20a], Microsoft [Jos20b] and Intel [GFS+19, CBB+18] are based on machine learning and in particular deep learning [LBH15, Sch15]. Nevertheless, when compared with biological systems, such algorithms spend considerable computational resources: energy, matter, time and space [CPC16]. All of which add to the economic and environmental burden. Furthermore, artificial intelligence remains distant from natural intelligence (see [Mar18] for a critical appraisal on deep learning and its relation to natural intelligence). This is not surprising; neither is it a duty of artificially intelligent systems to perform as biological ones in multimodal and complex knowledge structures or unpredictable real-world environments. Such abilities are, however, desirable. With the evolution of technology into pervasive computing, there was a paradigm shift on how humans relate with computers, as well as what we expect from or dream about them. This is, in the author's opinion, a more important motivation for post-von-Neumann architectures than the often cited end of Moore's law for miniaturisation [Pep17].

Biologically-inspired hardware architectures bring novel solutions aiming to both alleviate the computational cost, increase the cognitive aspects of computation, and

provide one of the means to improve our understanding of living matter at scales inaccessible experimentally. Here, we explore a disrupting technology known as neuromorphic computing [Fur16a, Mea90, LDI+14]. It is inspired by the most complex organ on earth, the brain. This latter is asynchronous and stochastic. Its computing units are distributed, decentralised, making it highly sparse in both space and time.These features are proposed to underlay its remarkable general intelligence and energy efficiency [DIPR07, HJM13, BS12], some theorists suggest that even the emergence of consciousness defined as intrinsic information with causal power in self-organizing agents. From these simple axiomatic definition, it is found that the underlying physical substrate of consciousness should have characteristics of complexity as those of the neocortex [TBMK16, Fri05]. In parallel, the main technological drivers for the study of the brain and brain-inspired machines are the development of novel computing paradigms and intelligent systems [AS15], and the understanding of ourselves with repercussion in the treatment of diseases and increase on the quality of life. Novel computing like neuromorphics, quantum computers and approximate computing paradigms have demonstrated good potential, but still, fail to demonstrate a wide range of abilities in real-world applications [WKM+19, NVS16, SPP+17, CRO+19, MV17, OML19, BM19, LGQO18]. One of our interest is to advance neuromorphic technology to a level where it could be clearly useful, covering a problem general enough to be transferable across disciplines. The other is to approximate neuromorphics to the medical and experimental neuroscience communities.

Structurally, the human brain is built from a high dimensional and very entangled structure, which has been challenging to unravel with the existing research methods [RNS+17, BS09]. Besides, its exploration is rightfully subject to ethical strict regulations and limitations on in-vivo and in-vitro studies. The number of neurons in the full human brain is in the order of 86 billion (note that non-neuronal cells also approximate 85 billion) [ACG+09, HH09]. Regarding connections, each neuron is able to form thousands of connections with other neurons by means of synapses, that could sum up to $\approx 10^{15}$ in the whole human brain [Mou57]. Despite its dimensionality and a high degree of randomness, the neural interconnect is neither purely random nor purely regular. It is instead a complex network. Complex networks present a nontrivial topology, emerging as a set of models that better describe the high dimensional systems found in nature. The graph topology of the brain preserves a structured and hierarchical connectivity [DM04, BS09, RNS+17], where only 19% of the brain neurons belong to the cerebral cortex, where intelligence and consciousness are thought to emerge [TBMK16] (see

however [ARTB19] for contributions of visceral processes to consciousness and cognition). The cortex is the outermost neuronal tissue in the brain, consisting of a folded and layered structure of 2 to 4 mm thick, where the number of layers depends on the cerebral region. Each layer has different neuronal composition and structural connectivity. The cortex is further divided into allocortex and neocortex, the latter –consisting of six layers– is the most evolved part of the human brain and accounts for $\approx 90\%$ of the neurons in the full cortex. Besides the layered structure, defined by the horizontal distribution of somata (the cell body of neurons), there is also a columnar organisation, emerging from a higher density of vertical connections across the thickness of the cortex, connecting the neurons of different layers that have similar tuning properties. Such cells form clusters that behave as microcircuits [Mou57][HW77]. An organisation that is involved in the neocortex ability to perform cognitive processes like memory, attention, perceptual awareness, language, thought, and consciousness [Abe91]. In the first part of this thesis, we harness one crucial aspect of the brain structure, the existence of a neural connectivity pattern which governs action selection under multiple choices, the so-called winner-takes-all (WTA) circuit, networks of WTAs allow the brain to combine acquired knowledge and input evidence from the external world in order to perform decision making [DM04, FSF$^+$13, JHM16]. Following [JHM16] and [RSD18], we show how such an organisation combined with stochastic dynamics across the network, a feature of brain dynamics, endow neuromorphic technology with the competitive ability for problem-solving. This latter is not only a necessary condition for intelligent behaviour, but it is also central to it [Rob16, Sar18, SBG$^+$15].

1.1.1 Neuromorphics for Problem-Solving

In the process of developing neuromorphic hardware, It is desirable to explore the capabilities of the machines on the realms of both general problems in mathematics and computational science, as well as on the understanding of cognition and animal behaviour. If the machine succeeds in representing or solving any of the well defined abstract classes of problems, it means it will apply to real-world problems that can be formulated under that formalism. The more general the class of problems, the broader the range of applications will be covered, as well as the better we will understand the capabilities of the design. More importantly, we will understand the limitations of it. Understanding the limits of applicability of a computational approach is as crucial as evolving its capabilities [Har04]. The reason is that some problems are indeed intractable. In the sense that, it does not matter how much we improve the speed,

power consumption or size of our computers, there are families of problems which despite being solvable with infinite resources, will remain intractable [HF04] (at least until some exotic machine demonstrates an exponential speed-up; quantum and genetic computers promise advances in this direction, at least in theory, but the practicalities seem to be out of the scope of our era). Even worse, are the undecidable or unsolvable problems, where infinite-time availability will not help. Thus, knowing the performance and complexity of a new computer architecture, in the hierarchy of computable and incomputable problems, will shed light on realistic directions for optimisation and improvement, avoiding the use of valuable time on aspects that will not add significant scientific or technological value.

For the first part of this thesis, I selected constraint satisfaction problems (CSPs) as a general problem with clear applications and which remains very hard to solve. It belongs to the NP-complete complexity class, for which the existence (or not) of efficient algorithms remains a major unsolved question in computational complexity theory [For09]. Thus, constraint satisfaction problems (CSPs) [DC$^+$03] are a special family of problems which serve the purpose of understanding the computational power and limitations of novel technology, having a wide range of applications and keeping connections with behaviour and cognition in neuroscience [DM04]. CSPs are beautifully simple to formulate, yet they seem to belong to the class of intractable problems. In particular to the NP-complete family. These are problems verifiable in polynomial time, sometimes understood as efficiently [Cob65, Edm65, GJ79], yet finding their solution requires supra-polynomial time on the size of the problem. The evidence suggests that the time-complexity may be exponential, e.g., a linear increase on the problem size results in an exponential growth on the required resources [For09].

Solving a CSP as an optimisation problem consists on travelling across a cost or energy hypersurface, defined on the high-dimensional space of all possible configurations, looking for a global energy minimum where the system is relaxed the most, i.e. satisfies a maximum number of constraints. Independently of the specific strategy used by an agent to solve the CSP, a fundamental limitation is that the agent will always have finite computational resources to perform the search. As the problem size increases, these resources quickly become insufficient for an exhaustive search. When this latter can be performed, the agent has full resolution information about the cost over the state space. However, for large problems the agent has to use sparse or scattered samples to perform the search, looking for a trade-off between high-resolution local information and low-resolution global information, The former guides a greedy search for optimal local

decisions. The latter is usually obtained from a discrete and finite sampling over the state space whose information content will decrease drastically with both, the curvature of the cost hypersurface and the dimensionality of the associated combinatorial space [DC$^+$03, Wal00].

Every solution to a CSP will have zero violations and include all variables defining the problem. Hence it will be represented by a global minimum of the cost hypersurface. If the problem has several solutions, the global minimum will be degenerated, existing one minimum for each solution. It is easy to see then, that the difficulty of finding a solution for a CSP is not only given by the high dimension of its combinatorial space. The curvature on such space is also critical. Here, the curvature refers to how folded is the cost function (a scalar function related to the number of unsatisfied constraints) in the space of possible evaluations. If the cost function is strictly convex, there will be a single minimum and methods like gradient descend will easily find it. However, this is rarely the case.

1.1.2 Neuromorphics for Neuron Modelling

Regarding the use of neuromorphics for brain modelling, given the experimental limitations in studying the different scales of the neural tissue, research must resort to computation as an alternative and fundamental method for accelerating neuroscience research, in other words, computers became the most practical option to explore the parameter space of the human brain. Even having full access to the properties of individual cells, it is highly probable that several questions should be answered at the ”mesoscale”, where millions of neurons are interacting by means of billions of synapses, an unfeasible experimental task that becomes natural for massively parallel neuromorphic computers. Due to the intrinsic complexity of the human brain, we should regard both the bottom-up and top-down strategies for its study. The bottom-up approach builds-up the system from its minimal components; however, nonlinearities become essential when the number of elements and interactions increase. The nonlinear dynamics cause the whole system to behave differently than just the sum of its components, if the system parameters reach some critical values, the emergence of phenomena like chaos and collective behaviour can happen, causing the bottom-up approach to losing its predictive ability. In such cases a top-down approach is worth exploring, In the example of the human brain, at the top, one analyses the cognitive processes and their interactions, and goes through cerebral regions down to the neurons. For some studies, this approach is challenging as well; for instance, due to the abstract nature of the top elements, the

measurements tend to be more qualitative than at the bottom level.

There is a general understanding of the function and behaviour of stereotypical neurons and the nature of their connections (synapses) [BG91]. Nevertheless, there is not a full characterisation of all types of neurons, which are largely diverse. Such diversity obscures the understanding of what is known as the neural code, how neurons operate from the perspective of information processing [DA+03]. To explore how neuromorphics could aid at modelling these diverse scales while harnessing data from experimental neuroscience, we established a collaboration with the Institute of Basic Medical Sciences of the University of Oslo with the long-term goal of porting one of the more accurate thalamo-cortical models developed by S. Hill and G. Tononi [HT05]. The recurrent activity between the cortex and and the thalamus are essential for the maintenance and regulation of sleep and depend on the interaction between several types of trans-membrane ion currents. Although the role of sleep remains a mystery, it represents a transition between conscious and unconscious states and a detailed account of the cellular mechanisms and electroencephalogram signatures that characterise sleep, wakefulness, as well as, the transition between these is available. Hill and Tononi, unified the most relevant experimental evidence in a single, multi-scale, anatomically and physiologically accurate computational model of the thalamocortical system which replicates the observed wakefulness, non-Rapid Eye Movement (REM) sleep and REM sleep states, their response to external visual stimuli and the transition between these states [HT05].

The implementation of this model in SpiNNaker required the implementation of several new functionalities in the machine. It uses a more complex network model which harness the advantages of both the point neuron models and the Hodgkin-Huxley model. In short, it required the implementation of:

- The pacemaker, low-threshold calcium, persistent sodium and depolarisation-activated potassium intrinsic ion currents [HM92]. This currents represent the passive contributions to the electrical properties of the neurons and emerge from the ion-channels distributed across the cellular membrane. This physiological feature was never implemented in neuromorphic hardware.

- A dynamic threshold which has a resting value, at each spike, it is reset to the sodium resting potential from where it decays exponentially towards the resting value [HT05]. The previous threshold in SpiNNaker was just a fixed value [RBB+18].

- NMDA voltage-gated conductances, these are voltage-sensitive synaptic conductances [VCR03].

- Short-term plasticity. This type of plasticity reflects the loss in performance of the signal transmission in the synapse when spikes are to close in time. It is due to the reduced availability of neurotransmitters and vesicles [ZR02].

The last chapter of this thesis is devoted to the implementation of generic voltage-gated ion-channel currents from which the pacemaker, low-threshold calcium, persistent sodium and depolarisation-activated potassium intrinsic currents underlying the Hill-Tononi model are a subset. Crucially, these intrinsic currents are responsible for triggering the wakefulness-sleep transition [HT05]. Our implementation endows the SpiNNaker machine with a neural dynamics which adapts according to the activity of the network and the cell. Opening possibilities for intrinsic plasticity and runtime neural adaptation. However, due to its fixed-point architecture, the implementation on SpiNNaker present some discrepancies with the floating-point implementations in conventional CPU hardware. Our work paves the road for the implementation of intrinsic currents in the next generation of the SpiNNaker machine which is currently under development.

1.2 Neuromorphic Hardware Platforms

This thesis is based on two state-of-the-art neuromorphic systems based on the SpiNNaker and Loihi chips respectively. The spiking neural network architecture (SpiNNaker), developed at The University of Manchester, is the largest neuromorphic computer ever build and the only supercomputer with one-million computing nodes (processor cores). The Intel Loihi Chip is the fastest neuromorphic chip to preserve a massive number of spiking neurons and some degree of model flexibility.

It has been widely acknowledged that computational neuroscience studies are faced by the principles of conventional parallel design, namely, memory coherence (through a global memory), full-synchronicity and determinism (in the sense of sequential computations) [Fur16a]. Such principles do not correspond with brain functioning, generating bottlenecks that needed to be overcome. Among others, SpiNNaker emerged to support real-time and efficient computing, as well as the ability to handle vast numbers of neurons and synapses. SpiNNaker was designed as a massively parallel supercomputer which disregards the classical paradigms, it allows an efficient multicast

and asynchronous global communication between distributed local dynamical systems describing neurons i.e., numerically-solved time-dependent differential equations. Communication happens through small messages, source-routed packets of 40 or 72 bits, representing action potentials or spikes. Notice the clear difference with conventional artificial neural networks. Spiking neural networks (SNNs) incorporate time, are recurrent and asynchronous, rather than synchronous feed-forward as many deep learning architectures [LBH15], literally opening a new dimension which brings both, opportunities for cognitive algorithms and challenges for developing them. Chapters 2 and 4 make a step forward in this regard.

Two versions of the SpiNNaker architecture exist. SpiNNaker 1 has been two decades in conception with half of this time devoted to its construction, while SpiNNaker 2 is in the prototype stage. Physically, the SpiNNaker machine is built from $\approx 10^6$ ARM9 processor cores, topologically arranged in a toroidal mesh. Each core has two tightly-coupled memories one of 32 Kbytes for instructions (ITCM) and one of 64 Kbytes for data (DTCM), the processors are built with hardware support for fixed-point arithmetic only, which saves power and silicon area. The cores are grouped in 5.7×10^4 18-core system-on-chip nodes. Each one with a multicast router and a 128 Mbyte off-die double data rate synchronous dynamic random access memory (DDR-SDRAM) shared by the 18 processors. Chips communicate through self-timed channels and are grouped on a printed circuit board forming 48-node hexagonal arrays, which then communicate via 3.1 Gbps high-speed serial interfaces, six per board. For further details on the SpiNNaker hardware see [FLP$^+$13, KLP$^+$08], the software is detailed in [RBB$^+$18, RBD$^+$18].

SpiNNaker 2 aims to upgrade approximately 100 times the already unique billion spiking neurons of its first generation, which corresponds to $\approx 1.15\%$ of the number of neurons in the human brain. This means that SpiNNaker 2 will be able to handle the same number of neurons as there are in the whole human brain, i.e. ≈ 86 billion leaky integrate and fire (LIF) neurons. Note that in both machines the number of neurons drops as biological details are included. Such an upgrade is achieved by substitution of the ARM 968 processors, manufactured with a 130 *nm* CMOS technology, with the state-of-the-art ARM M4F. These latter features a 22nm CMOS (GLOBALFOUNDRIES 22FDX) process [CMP$^+$16]. Furthermore, each chip is projected to contain 144 cores, 128K memory per core (which can be accessed synchronously by other cores) and a 2GB off-chip memory. Besides scaling, SpiNNaker2 incorporates improvements in energy efficiency. It maintains the $1\,W$ power consumption per chip, each of which integrates

the equivalent to a 48-node board from the first generation. Furthermore, through dynamic voltage and frequency scaling (DVFS) and adaptive Body biasing (ABB), the chip can switch between three (frequency-at-voltage) modes of power consumption according to workload [HSE+12] achieving an optimal power management [HYV+17].

With the above feature set, SpiNNaker is presented as a unique resource, with the striking features of performing real-time, asynchronous and event-driven high-dimensional simulations.

In turn, Loihi is a neuromorphic chip recently released by Intel corporation which differs from conventional computing architectures in similar design principles as the ones above for SpiNNaker. Because it brings a considerable speed-up for our CSP solver and has been known publicly for less time, a more detailed review of its software and hardware architecture is done in chapter 3.

1.3 Thesis Hypotheses

- Constraint satisfaction problems can be solved efficiently on neuromorphic hardware.
- Neuromorphic hardware can deliver competitive problem-solving with low energy expenditure and biologically relevant processing times.
- It is possible to validate CSP SNNs online (on-chip) without probing the neural activity of the entire network.
- Realistic postsynaptic currents are linearisable, allowing for their efficient implementation on neuromorphic hardware.
- Intrinsic currents are implementable in SpiNNaker

1.4 Thesis Contributions

1.4.1 Key Contributions

- Development of an application programming interface (API) for the solution of CSPs in the massively parallel SpiNNaker machine. The API constitutes an stochastic general-problem incomplete solver which requires a few seconds to

solve Sudoku and map colouring problems and converges to the problem solution for CSPs of moderate difficulty.

- Development of software and theoretical frameworks for the solution and on-chip validation of CSPs in Intel's Loihi neuromorphic chip and systems. The multicompartment spiking neural network which solves the problem also measure it's own cost function and notifies the host CPU when a solution to the problem is found. Only at this point, the solution is read from the network state. This methodology avoids constantly reading the network and transferring the data to host for post-processing, both of which cause an overhead of several minutes and seconds on SpiNNaker and Loihi respectively. On-chip validation implies satisfaction is known online. In our SpiNNaker solver, the solution and whether one was found are only known after gathering and processing the recorded network activity. The solver in Loihi achieves a speedup of two to three orders of magnitude improvement for Sudoku and map colouring problems (problems are solved in milliseconds).

- Development of a theoretical framework for the implementation in SpiNNaker of more biologically plausible alpha- and beta-shaped postsynaptic currents. SpiNNaker only supported Dirac delta and exponential kernels. These shapes control how synaptic input is integrated by a neuron into its membrane potential, directly affecting its spiking behaviour. The hard constraint is that required memory (buffer size and number of buffers) should not increase with the input spikes history. Thus, all previous activity has to be encoded on the last value in the memory local to the neuron.

- SpiNNaker implementation of voltage-gated ion-channel currents, also known as intrinsic currents. These follow the Hodgkin-Huxley formalism in which the neuron conductance is formulated in terms of the ensemble response of thousands of ion-channels across the neuronal membrane. The integration of data from electrophysiological recordings e.g., from voltage-clamp experiments on biological neurons, as well as their effect on the neuron dynamics, e.g., spike adaptation, is achieved. The intrinsic currents response to voltage changes remains within 10 % of the CPU double floating-point version, this error is expected due to the use of fixed-point arithmetic. Spike adaptation has qualitative correspondence with an equivalent implementation on the state-of-the-art neural simulator Nest. There are however quantitative differences on the exact spike times, all within a

few milliseconds, yet the same number of spikes is observed.

1.4.2 Secondary Contributions

- Demonstration of the applicability in diverse neuromorphic hardware of the framework presented in [JHM16] for solving CSPs with SNNs.
- Improvements in CSP solving times from those of SNN simulations [HJM13] and in CSP problem size over other neuromorphic implementations [BIP16, KM18].
- By achieving online solution to CSP problems in milliseconds our solver is in the timescale of conscious events of biological relevance, it can thus be considered as a realtime solver.
- An on-chip validation architecture which is applicable beyond CSP solvers to SNN architectures where certain subpopulations need to be controlled.
- The CSP solver was recently proposed as one of the benchmarks [Dav19] for progress in neuromorphic hardware.
- Energy-delay product competitiveness with the quantum annealers in solving CSPs.
- Increased the biological realism of simulations in the SpiNNaker Machine.
- Helped in bridging the gap between SNN simulations in neuromorphic hardware and experimental data acquired from biological neurons.
- Paved the way for the implementation of Hodgkin-Huxley neuron models in SpiNNaker 2.

1.5 Thesis Structure

The thesis is divided into two parts.

Part I. Problem-Solving from Network-level Stochastic Dynamics

This part deals with the endeavour of endowing neuromorphic hardware with the ability for problem-solving in the CSP class. I demonstrate the design, implementation and

solution of diverse CSPs on two-state-of-the-art architectures. We find promising results for thrusting neuromorphic technology into main-trend computing.

Chapter 2 **SpiNNaker Solver for Constraint Satisfaction Problems** This results chapter introduces CSPs, SNNs, the mapping of CSPs into SNNs, as well as their implementation into the SpiNNaker machine. The CSP solver achieves good performance and convergence to a solution. This chapter has been published with minor modifications in the journal Frontiers in Neuroscience [FGF17].

Chapter 3 **The Intel Loihi Neuromorphic Chip** This is a literature chapter were the Loihi hardware and software architectures are reviewed, highlighting the features of interest for the Loihi CSP solver presented in the subsequent chapter.

Chapter 4 **Loihi Solver for Constraint Satisfaction Problems with On-Chip validation** This results chapter demonstrates the design and implementation of an API for the solution of CSPs on Loihi. It features both offline (on-host) and online (on-chip) validation of satisfiability. The API I develop here enables the easy creation of diverse CSP applications.

Part II. Neuronal Excitability from Cellular-level Stochastic Dynamics

On this part, we improve the biological realism of neural models in SpiNNaker. Adding to their local complexity while keeping the scalability achieved through asynchronous event-driven computing. This section presents the possibility of parameterising SNNs with cellular data obtained from in-vitro and in-vivo measurements. Both postsynaptic and intrinsic currents emerge from stochastic dynamics at the molecular scale, our models however only capture the relevant average dynamics.

Chapter 5 **Neuromorphic Implementation of Postsynaptic Currents** This results chapter tackles the problem of the linearisation of the alpha and dual-exponential functions with stochastic spike arrival. The chapter demonstrates results for postsynaptic currents which drove their implementation on the SpiNNaker software tool-chain. The

derivations, however, enable the implementation of these functions to extend the eligibility traces used for plasticity and neuromodulation.

Chapter 6 **Intrinsic Currents Generated by Voltage-gated Ion Channels** This results chapter demonstrates the preliminary implementation of voltage-gated ion channel currents of biological interest, as well as their effect on the evolution of the neuronal membrane potential and firing pattern. Despite the limitations in accuracy by using fixed-point arithmetic, these results are adequate for simulations in which qualitative accuracy is desired. Further work is needed if high quantitative precision is required. This can, in principle, be achieved with SpiNNaker 2.

Chapter 7 **Conclusions and Future Work** This chapter closures the thesis presenting overall conclusions and directions for future work.

Part I

Problem-Solving from Network-level Stochastic Dynamics

Chapter 2

SpiNNaker Solver for Constraint Satisfaction Problems

[1]

Constraint satisfaction problems (CSPs) are at the core of numerous scientific and technological applications. However, CSPs belong to the NP-complete complexity class, for which the existence (or not) of efficient algorithms remains a major unsolved question in computational complexity theory. In the face of this fundamental difficulty, heuristics and approximation methods are used to approach instances of NP (e.g. decision and hard optimisation problems). The human brain efficiently handles CSPs both in perception and behaviour using spiking neural networks (SNNs), and recent studies have demonstrated that the noise embedded within an SNN can be used as a computational resource to solve CSPs [JHM16]. Here, we provide a software framework for the implementation of such noisy neural solvers on the SpiNNaker massively parallel neuromorphic hardware, further demonstrating their potential to implement a stochastic search that solves instances of P and NP problems expressed as CSPs. This facilitates the exploration of new optimisation strategies and the understanding of the computational abilities of SNNs. We demonstrate the basic principles of the framework by solving difficult instances of the Sudoku puzzle and of the map colour problem and explore its application to spin glasses. The solver works as a stochastic dynamical system, which is attracted by the configuration that solves the CSP. The noise drives the exploration of the space of configurations, looking for the satisfiability of all the

[1] This chapter has been published with minor modifications as a research article by the journal Frontiers in Neuroscience, Neuromorphic Engineering [FGF17] under the terms of the Creative Commons Attribution License (CC BY). The author of this thesis and his supervisor retain the copyright.

constraints; if applied discontinuously, it can also force the system to leap to a new random configuration effectively causing a restart.

2.1 Constraint Satisfaction Problems

Most practical problems and natural phenomena can be abstracted as systems composed of smaller elements interacting with each other, an element being able to assume one of many states and the global configuration of states governed by the nature of the interactions. In practice, each interaction imposes a restriction on the behaviour of the units (a constraint). Such a description allows the interpretation of the phenomena as a constraint satisfaction problem (CSP), which is defined by the tuple $\langle X, D, C \rangle$. Here, $X = \{x_1, ..., x_N\}$ is a set of N variables defined over the respective set of non-empty domains $D = \{D_1, ..., D_N\}$, each x_i represents an element of the system which can take $|D_i|$ possible states. The constraints $C = \{C_1, ..., C_m\}$ are $\langle S_i, R_i \rangle$ tuples defined over m subsets $S = \{S_1, ..., S_m : S_i \subseteq X\}$, and k relations $R = \{R_1, ..., R_k\}$ [RN16]. In general, each R_i is a tuple defined over the Cartesian product of the variable domains, if however, all relations R_i are defined as 2-tuples, the CSP is called binary. With this definition, and without taking into account symmetry considerations, one has on the order of $\overline{D}^N$ possible evaluations for the values of the set X. (Here $\overline{D}$ is the average size of the domains). In the case of a Sudoku puzzle, for example, X represents the grid cells, the set D consists of the nine possible digits for each cell, and C defines the game rules. In this case one has 9^{81} possible configurations which after puzzle equivalency reduction define $\approx 6.67 \times 10^{21}$ possible puzzles [FJ05a].

A solution to the CSP (if it exists) is a valuation of X that is consistent (satisfies all the constraints C_i in C) and complete (includes all variables x_i in X). To find such a solution, one implements a search algorithm that explores the state space of all these configurations [DC$^+$03]. The strategy of searching the whole state space, known as the brute-force algorithm, quickly becomes unfeasible as N increases (e.g. requiring more computing time than the age of the universe [Nor09]), demanding the development of cleverer algorithms [DC$^+$03]. The efficiency of such computing algorithms is conventionally determined with the definition of its asymptotic time complexity $T(n)$, expressed as a function of the input size of the problem $n \propto N$ for a particular encoding language [GJ79]. Notice that for a given problem two different instances of the same size n could reveal different performance, so T refers to the worst-case complexity. According to Cobham's thesis [Cob65, Edm65], an algorithm is

conventionally considered efficient if it admits worst-case polynomial-time solutions on a deterministic Turing machine (DTM). Such algorithms build up the P complexity class, corresponding to $T(n) \in O(n^{\kappa})$, where κ is determined by the nature of the problem. A broader class, the NP complexity, contains all decision problems for which a proposed solution can be verified in polynomial time [Coo71].

The problem of determining the existence of efficient algorithms for solving every NP problem, known as the P versus NP problem, remains unsolved since its establishment by [Coo71]. When a problem does demand algorithms outside P, it is said to be intractable, and it is a widely held view that this is the case for a large subset of NP. Thus, instances of NP are recognised as very hard problems [For09], the hardest of which is referred to as NP-Complete, which are NP problems to which any other NP problem can be reduced in polynomial time, hence completeness [Kar72]. [2] If $P \neq NP$, NP-complete problems are tractable only by an ideal non-deterministic version of the Turing machine (NDTM) [Coo71, Kar72, GJ79]. We can think of Turing machines as abstract devices endowed with a set of rules to act on a string of symbols, such actions depending on both, the machine's internal state(s) and the input symbol(s). While at each computation node a DTM has a specific action to perform (thus defining a computation path) an NDTM can follow a whole family of actions (thus defining a computation tree) [HMU06]. At each computation step, either the NDTM takes action biased towards configurations that lead to accepting states or it branches executing all of the allowed actions [Mar11]. In any case, an NDTM is guaranteed to find a solution if it exists. Although the biased action description is unrealistic, the replicative interpretation is only limited by the available space and time resources (increasing resources are needed as the NDTM advances through the computation tree). Despite the apparent impracticability of manufacturing an NDTM, very recently, and based on the replicative properties of the deoxyribonucleic acid (DNA) molecule, [CKA$^+$17] reported the first physical design of the embodiment of an NDTM. The practicability of NDTM remains, however, uncertain in the near future. Therefore, with a high possibility of $P \neq NP$ and no NDTMs available, NP problems stay as a hard task to be tackled. Importantly, the determination of the existence (or not) of solutions for a CSP constitutes an NP-complete problem. Therefore,

1. there are no known efficient algorithms that work for general CSPs, despite the fact that there are polynomial-time subcases and

[2] A set of yet harder problems form the NP-Hard class of which P, NP and NP-Complete problems are subsets, though NP-Hard problems are not necessarily NP.

2. any other NP problem can be expressed as a CSP in polynomial time.

2.2 Neuronal Approaches to Constraint Satisfaction

NP-Complete problems find applications in a wide range of fields, from spin-glass systems, resources allocation and combinatorial mathematics, to Atari games and public-key cryptography [GJ79, For09, Bar82, ADGV15]. Thus, in the absence of known efficient algorithms for solving general NP problems, and the need for at least an approximate solution, the standard strategy is to find either an adequate heuristic or an approximation algorithm for the particular instances of the given problem. Such non-neural strategies have been successful for developing practical applications [DC$^+$03]. Here, we instead take inspiration from the underlying neuronal network architecture which biological organisms use to efficiently cope with CSPs, in this case even the limitations found are enlightening, i.e. it could be more convenient for an animal to prioritise a nearly-optimal but quick solution, especially if the system is unsolvable. In [HT85], Hopfield, J. J. and Tank, D. W. firstly proposed stochastic analogue neural networks to solve decision and optimisation problems. They had realised the CSP nature of their previously implemented content addressable memory [Hop82], and of the optimisation of perceptual inference by [HS83], both of which used networks of binary neurons. [3]. More recently, an alternative approach based on deterministic multistable neural oscillators and synaptic plasticity was proposed [MMI13]. All the neural models above are liable to get stuck in local minima. This problem was tackled by [MMI15b] by enhancing the model of [MMI13] with the use of gamma-band rhythmic oscillations of incommensurable frequencies (not rational multiples of each other), which further allowed their network dynamics to stabilise when all constraints are satisfied. The latter gave rise to an event-driven, mixed analogue/digital prototype chip of incommensurable oscillators which, bespoken to the distributed nature of CSPs, promises to yield state-of-the-art performance [MMI15a].

In the middle of the 90s, more biologically plausible versions of neural networks, the SNNs, were demonstrated to present equal or superior computational capabilities than those of analogue neurons [Maa95, Maa96, Maa97]. Despite promising advantages,

[3]The main difference between Hopfield nets and SNNs for solving CSPs is the asymmetry in the ON and OFF state of the neurons, Hopfield's approach uses analogue neurons which keep the CSP network state well defined, in contrast, stochastic spiking neurons have an inherent asymmetry between their random spiking probability and their deterministic reset after a spike. This feature gives SNNs the possibility of bypassing energy barriers in the stochastic search because the CSP state can be temporarily undefined [JHM16]

their implementation demands a high computational expense in conventional hardware. Regarding CSPs, [MB00] achieved an SNN solution of an eight cities travelling salesman problem (TSP). More than a decade later, [HJM13] demonstrated that the stationary distribution of a stochastic SNN visits the solution of a hard Sudoku puzzle on average 2% of the time once it acquires a performance where 90% of the constraints are satisfied, and finally [JHM16], formalised the application of SNNs to general CSPs, postulating a methodology which allows the shaping of the energy landscape, using a modularity principle, controlling the network dynamics and causing it to visit the solution to the problem with high probability.

2.3 Neuromorphic Approaches to Constraint Satisfaction

The models above suggest that the noisy, distributed and asynchronous nature of the brain's processes could be behind its computational properties, contrasting with the conventional trends in commercial computer architectures. The brain itself is constantly facing conflicting situations where it should decide actions that best satisfy a number of constraints [Chu08, Tom15]. Hence, we can take advantage of the brain-inspired computers (neuromorphics) to design new strategies for solving CSPs and gain understanding about which of such strategies are biologically plausible. Given the NP-complete nature of CSPs, it seems natural to consider the research on SNN-solvers to be at an early stage, with the need for an even deeper exploration of their dynamics. It is the aim of this chapter to provide a tool for the exploration of the behaviour of high-dimensional networks running in biological real-time, facilitating the further evolution of SNN-solvers for CSPs, allowing, for example, the study of the non-Boltzmann and non-Markovian dynamics of the network [CB90, CFT$^+$15]. For this, we use the Spiking Neural Network Architecture (SpiNNaker), a neuromorphic computer which presents a nice balance between the very large number of neurons it is able to simulate, its energy efficiency and the biological real-time feature of the simulations. Neuromorphic computers are electronic devices emulating the working mechanisms of the brain in the search for alternative models of computation. They aim to overcome the limitations offered by conventional computational architectures especially (but not only) with regard to brain simulations [Mea90, LDI$^+$14, Fur16a, Fur16b]. Similarly to the prototype chip of incommensurable oscillators of [MMI15a], neuromorphics provide a distributed architecture that resembles that of CSPs. They

also share the local nature of the constraint graph in which generally a constraint relate only a few variables. SpiNNaker is a real-time asynchronous, multicast, and event-driven machine [FLP+13, FGTP14], features that favour the implementation of stochastic computations. Furthermore, it is designed to compute with spiking neurons, overcoming the computational cost that historically limited implementations of SNNs compared to artificial neural networks. Through the following sections, we are going to show how SpiNNaker is able to implement a stochastic search that solves constraint satisfaction problems. Besides running in biological time, our approach improves previous stochastic SNN implementations with the ability to converge into a stable (long-lasting) solution.

2.4 From Constraint Satisfaction Problems to Spiking Neural Networks

In order to implement the stochastic search, we first need to map our CSP into an SNN. Formally, a spiking neural network can be defined as a set of spiking neurons $\mathcal{N}$, each one with a threshold function θ_i, and with connections between two arbitrary neurons $\mathcal{N}_i$ and $\mathcal{N}_j$ established by the set of synapses $\mathcal{S} \subseteq \mathcal{N} \mathrm{X} \mathcal{N}$. For each element $\mathcal{S}_{i,j} \in \mathcal{S}$ there is a weight parameter $w_{i,j}$ and a response function $\mathcal{R}_{i,j} : \mathbb{R}^+ \rightarrow \mathbb{R}$ [Maa97]. In our implementation each neuron $\mathcal{N}_i$ corresponds to a leaky integrate and fire (LIF) neuron [Ste67]. In this model the dynamics of the membrane potential v_i are given by:

$$\tau_m \frac{dv_i}{dt} = -v_i(t) + RI(t). \tag{2.1}$$

Here, τ_m is the membrane time constant, R is the membrane resistance and I an external input current. Each time v reaches a threshold value θ a spike is elicited; such events are fully characterized by the firing times $\{t^f \mid v(t^f) = \theta \, and \, \frac{dv}{dt}|_{t=t^f} > 0\}$. Immediately after a spike the potential is reset to a value v_r, such that $\lim_{t \rightarrow t^{f+}} v(t) = v_r$. In our network, synapses are uniquely characterized by w_{ij} and the inter-neural effective separation that a spike travels, from the presynaptic axon hillock to the synapse, is introduced through an axonic time delay Δ_{ij}. In biological neurons, each spike event generates an electrochemical response on the postsynaptic neurons characterised by $\mathcal{R}_{i,j}$. We use the same function for every pair (i, j), this is defined by the exponential

postsynaptic current:

$$J(t) = \frac{q}{\tau} e^{-\frac{t-t_0}{\tau}} \Theta(t-t_0), \tag{2.2}$$

where q is the total electric charge transferred through the synapse, τ is the characteristic decaying time of the exponential function, $t_0 = t^f + \Delta_{ij}$ is the arrival time of the spike and Θ represents the Heaviside step function. The choice of $\mathcal{R}_{i,j}$ potentially affects the network dynamics, the use of the exponential function in the equation 2.2 constitutes one of the differences of our solver and the previous studies on solving CSP through SSNs which used a simple square function. In chapter 4, we will also use exponential PSPs for the solver with off-chip validation but will resource to square PSPs in the neural architecture for on-chip validation as these are part of the underlying theoretical proofs of convergence to a stationary distribution by [Jon14] and such PSPs are available in Loihi.

In an SNN each neuron is part of a large population. Thus, besides the background current $I(t)$, the principal neurons here receive input from the other neurons, as well as a stochastic stimulation from noisy neurons implementing a Poisson process. These latter emulate the brain's background activity. In this case, the temporal evolution of the membrane potential (equation 2.1) generalises to:

$$\tau_m \frac{d}{dt} v_i = -v_i(t) + R \left[I(t) + \sum_j w_{i,j} \sum_f J(t - t_j^f) + \sum_k \Omega_i J(t - T_k) \right] \tag{2.3}$$

where the index f accounts for the spike times of principal neuron j, Ω_i is the weight of the connection between the i neuron and its presynaptic Poisson spikes source which send random spikes which occur at time T_k, and $J(.)$ is the response function of equation 2.2. An SNN has the advantage that its microstate $\psi_t = \{n_1, n_2 ..., n_{|\mathcal{N}|}\}$ at any time t can be defined by the binary firing state $n_i \in \{0,1\}$ of each neuron $\mathcal{N}_i$, instead of their continuous membrane potential $v_i \in \mathbb{R}$. Then, the set of firing times $\{t_i^f\}$ for every neuron $\mathcal{N}_i$, or equivalently the set of states $\{\psi_t\}$, corresponds to the trajectory (dynamics) of the network on the state space. The simulations in this work happen in discrete time (time step = $1\,ms$), so in practice, ψ_t defines a discrete stochastic process (e.g. a random walk). If the next network state $\psi_{t_{i+1}}$ depends on ψ_{t_i} but is conditionally independent of any ψ_{t_j} with $j < i$, the set $\{\psi_t\}$ also corresponds to a Markov chain. In [HJM13], Habenschuss S. et. al. demonstrated that this is the case when using rectangular PSPs and a generalised definition of the network state, the validity of the

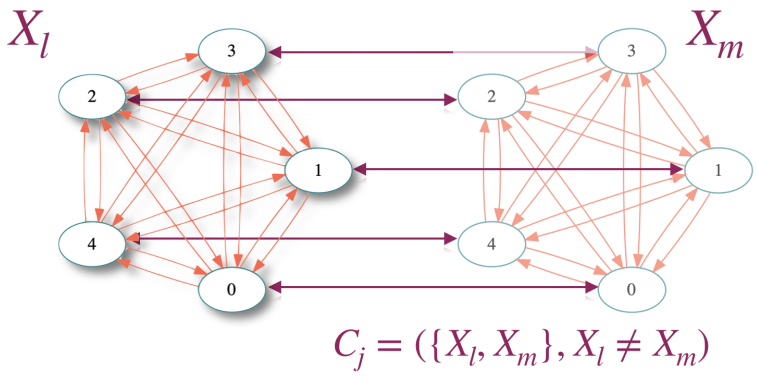


Figure 2.1: Competing WTAs encoding a non-equal constraint between two variables with 5 possible values each.

Markov property for general SNNs could still depend on the dynamical regime and be affected by the presence of a non-zero probability current for the stationary distribution [CB90]. Each possible configuration of the system, a microstate ψ_i, happens with certain probability p_i and, in general, it is possible to characterise the macroscopic state of the network with the Shannon entropy (in units of *bits*) [Sha48]:

$$S = -\sum_i p_i \log_2(p_i) \tag{2.4}$$

and the accumulated network activity:

$$A(t) = \frac{1}{|\mathcal{N}|} \sum_j^{|\mathcal{N}|} \sum_f \delta(t - t_j^f) \tag{2.5}$$

To compute p_i and hence equation 2.4 we binned the spikes from each simulation with time windows of 200 *ms*, notice in figures (2.2, 2.4, 2.5) how the activity of the networks in this chapter produces hundreds of spikes per *ms*, so windows of the order of 10^2 *ms* result in a sample size in the order of 10^3 spikes which is the same order of magnitude as the number of principal domain populations, making it representative of the network activity. In this type of high-dimensional dynamical system, sometimes the particular behaviour of a single unit is not as relevant as the collective behaviour of the network, described for example by equations 2.4 and 2.5.

SNNs can encode CSP variables and constraints as shown in figure 2.1, on it the variables X_l and X_m can take one of five possible values, each value is represented by a neuron and the five neurons inside each variable are competing through lateral

inhibition. An *All-Different* constraint $C_j = (X_l, X_m, x_l \neq X_m)$ can then be encoded as one-to-one inhibition between neurons in different variables encoding the same values. A complete CSP can now be encoded as an SNN with the pseudo-code of algorithm 2.1. This follows three basic steps: a) create SNNs for each domain D_i of each variable, every neuron is then excited by its associated noise source, providing the necessary energy to begin the exploration of the states $\{\psi\}$. b) create lateral-inhibition circuits between all domains that belong to the same variable. c) create lateral-inhibition circuits between equivalent domains of all variables appearing in a non-equal constraint and lateral-excitation circuits for domains in a all-equal constraint. With these steps, the resulting network will be a dynamical system representation of the original CSP. Different strategies can now be implemented to enforce the random process over states ψ_t to find the configuration ψ_s that satisfies all the constraints. The easiest and proposed way of implementing such strategies is through the functional dependence of the noise intensity with time. The size of each domain population n acts as a hyperparameter which should be large enough to average out the stochastic spike activity. Otherwise, the system will not be stable and will not represent quasi-equilibrium states. Also, n should be the minimum value that satisfies this criterion to save in neurons count and hence use less energy. As will be shown, the size of the domain populations allows the SpiNNaker solver to converge into a stable solution. This is, however, not a requirement in chapter 4 arguably because of the nature of the noise injection, which there consist in randomizing v rather than in stochastic spike trains.

The ensemble of populations assigned to every CSP variable x_i works as a winner-take-all circuit through inhibitory synapses between domain populations, which tends to allow a single population to be active. However, the last restriction should not be over-imposed, because it could generate saturation i.e., the inhibition from the initial winners is excessively strong to allow the network to swap valuations, and our network will be trapped in a local minimum. Instead, the network should constantly explore configurations in an unstable fashion converging to equilibrium only when satisfiability is found. The random connections between populations, together with the noisy excitatory populations and the network topology, provide the necessary stochasticity that allows the system to search for satisfiable states. However, this same behaviour traps some of the energy inside the network. For some problems, a dissipation population could be created to balance the input and output of energy or to control the entropy level during the stochastic search. In general, there may be situations in which the input noise acquired through stimulation can stay permanently in the SNN. Thus,

the inclusion of more excitatory stimuli will saturate the dynamics in very high firing rates, which potentially reaches the limit of the SpiNNaker communication fabric. In these cases, inhibitory noise is essential too and allows us to include arbitrarily many stimulation pulses. We will see in the next section that this simple approach provides an effective strategy for finding solutions to the studied CSPs through stochastic searches.

Listing 2.1: Translation of a CSP into an SNN.

```
# define the CSP = <X,D,C> through a set of lists.
    X=list(variables)
    D=list(domains)
    S=list(subsets_of(X))
    R=list(relations_over(s_i in S))
    C=list(constraints = tuple(s_i,r_i))
#a) create an SNN for each variable with sub-populations for each
    domain.
    n = size_of_ensemble, chose the minimum which still results in convergence.
    for variable x_i in X:
        for domain d_i in D:
            population[x_i][d_i] = create an SNN with n neurons
            noise_exc[x_i][d_i] = create a set of noise
            stimulation populations.
            apply_stimuli(noise[x_i][d_i], population[x_i][d_i])
            noise_inh[x_i][d_i] = create a set of noise
            dissipation populations.
            apply_dissipation(noise_inh[x_i][d_i], population[x_i][d_i])
#b) use inhibitory synapses to activate, on average, a single domain per
    variable
        for domain d_i in D:
            for domain d_j in D
                inhibitory(population[x_i][d_i],population[x_i][d_j])
#c) map each constraint to an inhibitory or excitatory synapse.
    for constraint c_i in C:
        read subset s_i and relation r_i from c_i
        for variables x_i and x_j in s_i:
            for domain d_i in D:
                if constraint relation r_i <0:
                    inhibition(population[x_i][d_i], population[x_j][d_i])
                elif constraint relation r_i >0:
                    excitation(population[x_i][d_i], population[x_j][d_i])
```

2.5 The Spiking Neural Network Architecture (SpiNNaker)

With large CSPs the equivalent SNN becomes computationally too expensive for conventional computers, so one of the important contributions of our work is the implementation of the SNN-solver on a computer architecture specially designed for computations

with spiking neurons. Conventional supercomputers physically embody a deterministic universal Turing machine and are designed to do computations transferring a high quantity of data in deterministic, synchronous, repeatable and reliable ways. Although under specif circumstances neuromorphic computers can be described by a DTM. They are devices inspired by the working principles of the brain, which is rather asynchronous and unreliable and thus has additional features. Although conventional machines have achieved impressive performance in automatic computing tasks –in part due to the great progress in miniaturisation– when facing the complex inference and cognitive tasks solved naturally by living organisms, biology outperforms them by several orders of magnitude, especially with regard to energy efficiency. We believe that such features can provide advantages in the solution of unsolved problems such as the ones in NP.

Neuromorphic computing was first introduced by Carver Mead in the 1980s, originally intended for analogue very-large-scale integration systems. Almost 30 years after Mead's work and after a decade of parallel efforts, there are but a few very powerful, massively parallel neuromorphic computers: TrueNorth [MAAI$^+$14], Neurogrid [BGM$^+$14], BrainScaleS [SBG$^+$10], Loihi systems [DSL$^+$18] and SpiNNaker [PPG$^+$13]. The latter is endowed with the ability to model high-dimensional spiking neural networks, low energy requirements, and a multicast communication protocol. It is based on a globally asynchronous and locally synchronous (GALS) multi-core System-on-Chip, being event-driven and able to run in biological time. SpiNNaker is built using a million ARM 968 processor cores. Each chip on the machine includes 18 processor cores connected by a network on chip (NoC) communication system [FGTP14, FLP$^+$13, PPG$^+$13, Fur12, GKK$^+$13, GF11]. This fundamentally different architecture paradigm, besides bespoke design for neurobiology simulations, makes the SpiNNaker system interesting for exploring new implementations of stochastic searches. Here we explore the computing power of the machine for these more general computing problems, exploiting the neuromorphic's ability to overcome the conventional difficulties of dealing with computationally expensive spiking neurons when implemented on conventional clusters and GPUs. In summary: i) for SpiNNaker spiking neurons are the fundamental modelling units and ii) it is a machine intrinsically able to implement stochastic computations on hardware. We will show in the next section how these two features bring new opportunities to solve hard constraint satisfaction problems.

2.6 Solving CSPs on SpiNNaker

In order to demonstrate the implementation of the SNN solver, SpyNNCSP, we present solutions to some instances of NP problems. Among the NP-complete problems, we have chosen to showcase instances of graph colouring, Latin squares and Ising spin glasses. Our aim is to offer a tool for the development of stochastic search algorithms in large SNNs. We are interested in CSPs to gain an understanding about the dynamics of SNNs under constraints, how they choose a particular state and their computational abilities. Ultimately, SNNs embedded in neuromorphic hardware are intended for the development of new technologies such as robotics and neuroprosthetics, constantly interacting with both the external devices and the environment. In such applications the network needs to adapt itself to time-varying constraints taking one or multiple decisions accordingly, making the advancement in stochastic searches with SNNs a fundamental requirement for neuromorphics.

2.6.1 Graph Colouring

Considering a graph G defined by the ordered pair $\{V,E\}$, with V a set of vertices and E the set of edges connecting them, the graph colouring problem consists of finding assignments of k colours to the elements of the graph (either V, E or both) such that certain conditions are satisfied [Dai80]. In vertex colouring, for example, the colours are assigned to the elements of V in such a way that no adjacent nodes (those connected by an edge) have the same colour. A particularly useful application of this problem is the process of register allocation in compiler optimisation which is isomorphic to graph colouring [Cha82]. Regarding time complexity, the general graph colouring problem is NP-complete for $k > 2$. In the case of planar graphs, 3-colouring is NP-complete and, thanks to the four colour theorem proved by Kenneth Appel and Wolfgang Haken, 4-colouring is in P [AH89].

A division of a plane into several regions can be represented by a planar graph, familiar versions of which are the geographic maps. In figure 2.2A we show the SNN-solver result of a satisfying 4-colouring of the map of the world where colours are assigned to countries such that no bordering countries have the same colour. We have followed the list of countries and borders from the United Nations available in Mathematica Wolfram [WR17]. The corresponding connectivity graph of the world map in figure 2.2A is shown in figure 2.2B. The insets in figure 2.2A, show the results of our solver for 3-colouring of the maps of the territories of Australia (bottom-right) and

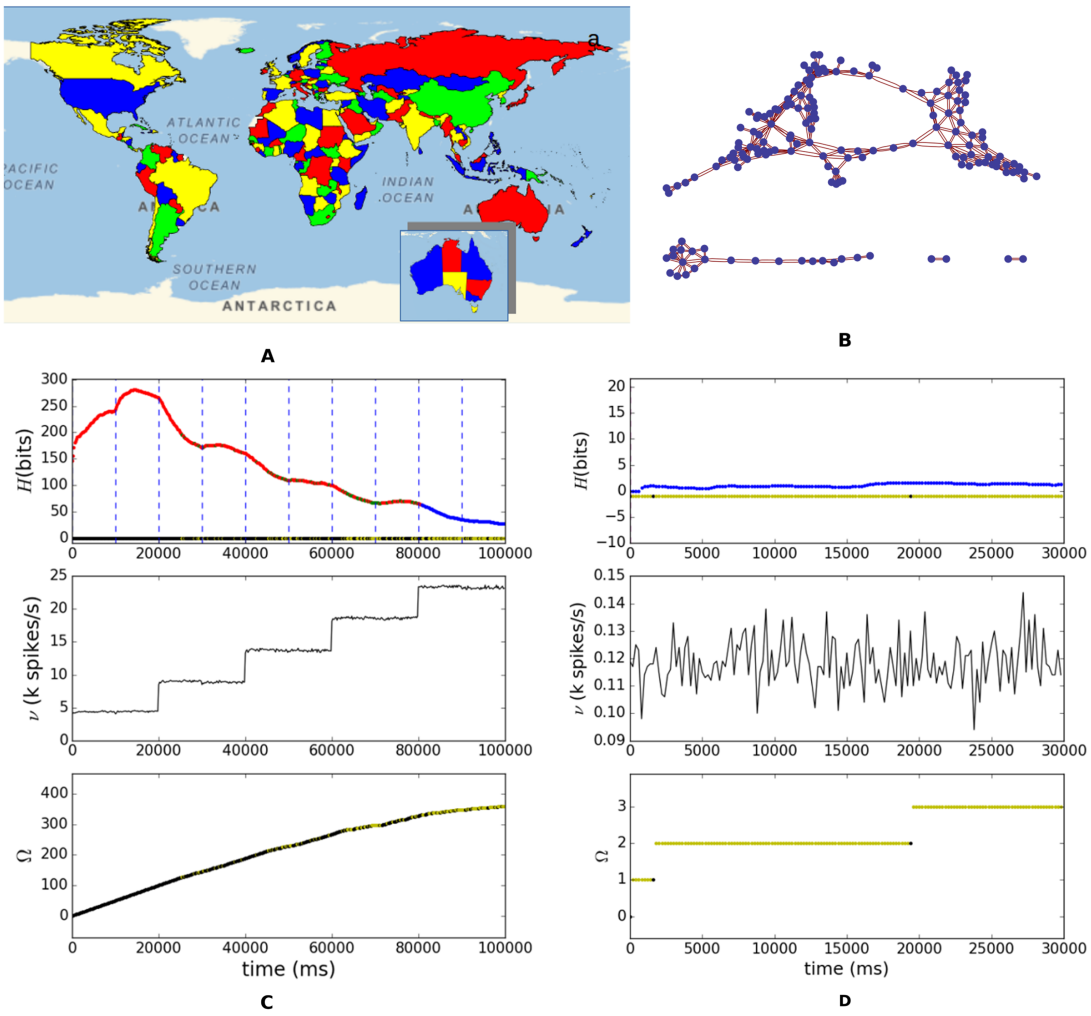


Figure 2.2: (A) Solution to the map colouring problem of the world with 4 colours and of Australia and Canada with 3 colours (insets). (B) Shows the graph of bordering countries from (A). The plots of the entropy H (top), mean firing spike rate ν (middle) and states count Ω (bottom) v.s. simulation time are shown in (C) and (D) for the world and Australia maps, evidencing the convergence of the network to satisfying stationary distributions. In the entropy curve red codes for changes of state between successive time bins, green for no change and blue for the network satisfying the CSP. In the states count line, black dots mean exploration of new states; the dots are yellow if the network returns to states visited before.

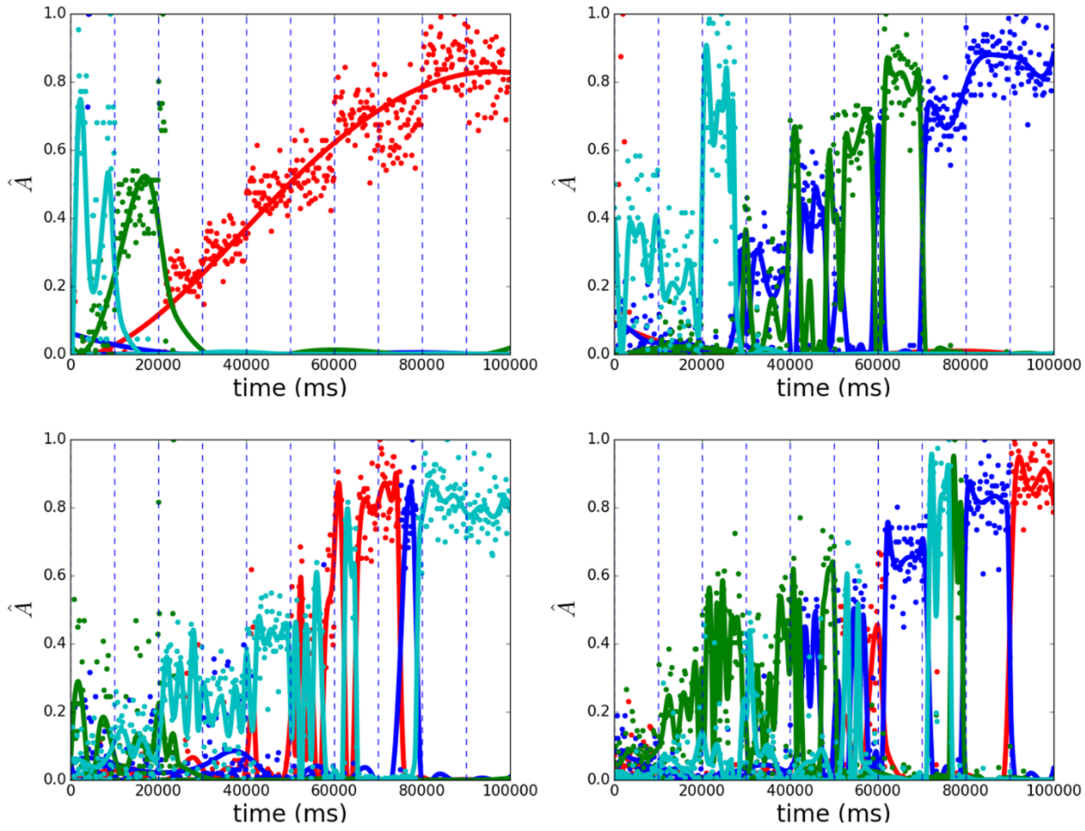


Figure 2.3: Population activity for four randomly chosen CSP variables from figure 2.2 (A), each line represents a colour domain.

of Canada (top-left). Figure 2.2C and 2.2D show the time dependence of the entropy (top), firing rate (middle) and the number of visited states (bottom) for the map of the world and of Australia respectively. The colour code we use in these, and the following figures is as follows: red means that the state in the current time bin is different from the one just visited, green represents the network staying in the same state and blue means that all constraints are satisfied. The dashed vertical lines mark the times at which noise stimulating (blue) or depressing (red) populations began to be active. The normalised spiking activity of the four colour populations for four randomly selected countries of the world map is shown in figure 2.3 evidencing the competing behaviour along the stochastic search. Interestingly, although the network has converged to satisfaction during the last $20\,s$ (blue region in 2.2C), the bottom right plot in 2.3 reveals that due to the last stimulation the network has swapped states preserving satisfaction, evidencing the stability of the convergence. Furthermore, it is noticeable in 2.2D that new states are visited after convergence to satisfiability, this is due to the fact that, when multiple solutions exist, all satisfying configurations have the same probability of happening. Although we choose planar graphs here, the SNN can implement any general graph, hence more complicated P and NP examples could be explored.

2.6.2 Latin Squares

A Latin square is defined as an array of $n \times n$ cells in which n groups of n different symbols are distributed in such a way that each digit appears only once in each row or column. The NP-completeness of completing a partially filled Latin square was demonstrated by [Col84], and among the useful applications of such a problem, one can list authentication, error-detection and error-correction in coding theory. Here we choose the Sudoku puzzle as an instance of a Latin square, in this case, $n = 9$ and in addition to the column and row constraints of Latin squares, Sudoku requires the uniqueness of the digits in each 3×3 sub-grid.

We show in figure (2.4) the solution to an easy puzzle [ERT12], to a hard Sudoku [HJM13] and to the AI Escargot puzzle which has been claimed to be the world hardest Sudoku. The temporal dependence of the network entropy H, firing rate ν and states count Ω is shown in figures 2.4A-C respectively for the easy (2.4G), hard (2.4H) and AI escargot (2.4I) puzzles. In figure 2.4E we show a schematic representation of the dimensionality of the network for the easy puzzle (G), each sphere represents a single neuron, and synaptic connections have been omitted for clarity, the layer for digit 5 is represented also showing the inhibitory effect of a single cell in position (1,3) over

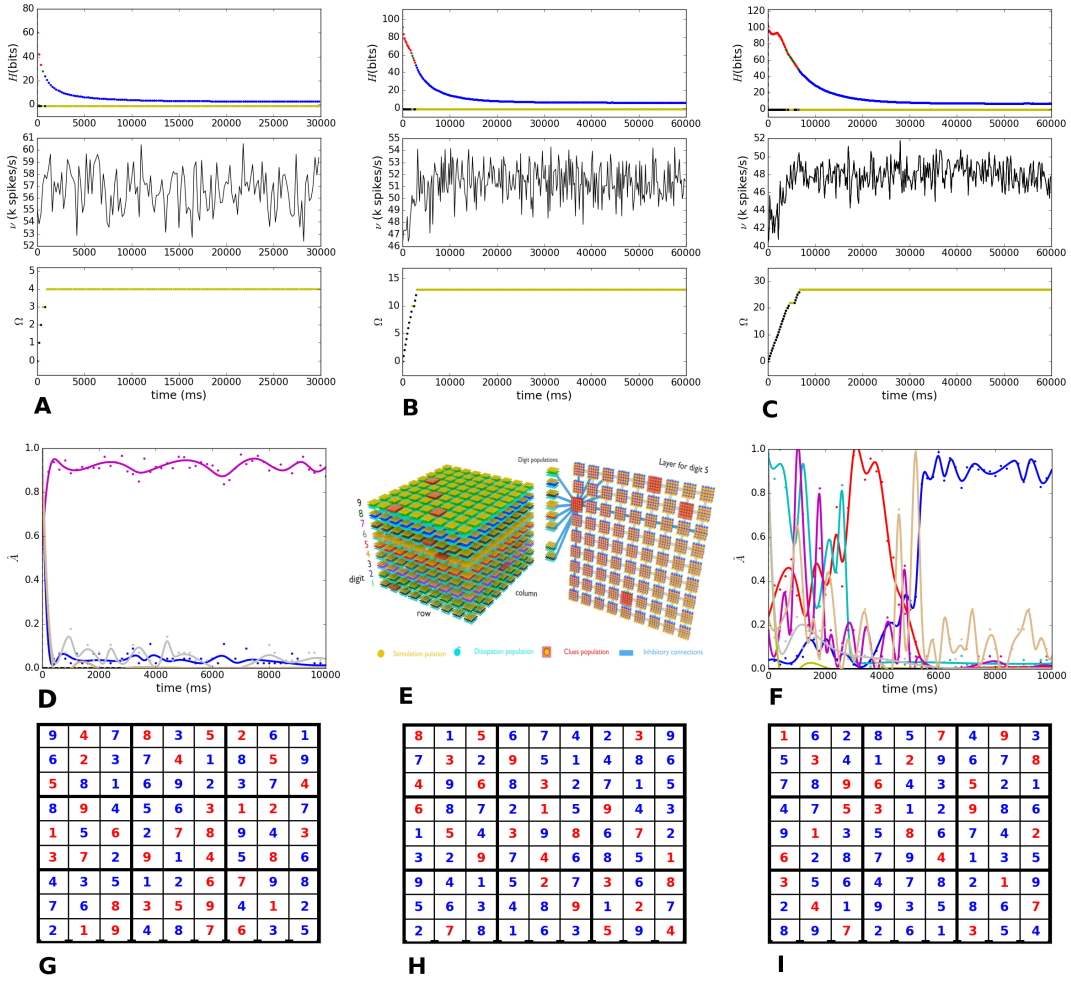


Figure 2.4: Spiking neural network solution to Sudoku puzzles. (A-C) show the temporal dependence of the network entropy H, firing rate ν and states count Ω for the easy (G), hard (H) and AI escargot (I) puzzles. The colour code is the same as that of figure 2.2. In (G-I) red is used for clues and blue for digits found by the solver. Figures (D) and (F) illustrate the activity for a randomly selected cell from (A) and from (C) respectively, evidencing competition between the digits, the lines correspond to a smoothing spline fit. (E) schematic representation of the network architecture for the puzzle in (A).

its row, column, subgrid and other digits in the cell. In this case, the total number of neurons is $\approx 37k$ and they form $\approx 86M$ synapses.

One improvement of our implementation with respect to previous work on SNNs [HJM13], is the convergence to a stable solution, the use of sub-populations instead of single neurons to represent the domains of the CSP variables was required to provide such stability to the network. The use of the more realistic exponential postsynaptic potentials instead of the rectangular ones used in the proofs of [HJM13] is also reflecting a good performance of the search as shown in the bottom plots in figures 2.4A, 2.4B and 2.4C, where the solution is found after visiting only 3, 12 and 26 different states and requiring $0.8\,s$, $2.8\,s$ and $6.6\,s$ respectively, relating well also with the puzzle hardness. It is important to highlight that the measurement of the difficulty level of a Sudoku puzzle is still ambiguous and our solver could need more complex strategies for different puzzles, for example in the transient chaos-based rating of [ERT12] the "platinum blonde" Sudoku is rated as one of the hardest to solve, and although we have been able to find a solution for it, it is not stable, which means one should control the noisy network dynamics in order to survive the long escape rate of the model presented by [ERT12]. We show in figures 2.4D and 2.4F the competing activity of individual digit populations of some randomly chosen cell in both the easy and the AI escargot puzzles, the dynamic behaviour resembles that of figure 2 in [ERT12] when comparing their dynamic solver for this same easy puzzle and the platinum blonde. Further analysis would bring insights into the chaotic dynamics of SNNs when facing constraints.

2.6.3 Ising Spin Systems

For each atom that constitutes a solid, it is possible to define a net spin magnetic moment $\vec{\mu}$ which results from the intrinsic spin of the subatomic particles and the orbital motion of electrons around their atomic nucleus. Such magnetic moments interact in complex ways giving rise to a range of microscopic and macroscopic phenomena. A simple description of such interactions is given by the Ising model, where each $\vec{\mu}$ in a crystal is represented by a spin $\vec{S}$ taking values from $\{+1,-1\}$ on a regular discrete grid of points $\{i,j,k\}$. Furthermore, the interaction of the spins $\{\vec{S}_i\}$ is considered only between nearest neighbours and represented by a constant $\mathcal{J}_{i,j}$ which determines if the two neighbouring spins will tend to align parallel $\mathcal{J}_{i,j} > 0$ or anti-parallel $\mathcal{J}_{i,j} < 0$ with each other. Given a particular configuration of spin orientations ω, the energy of the

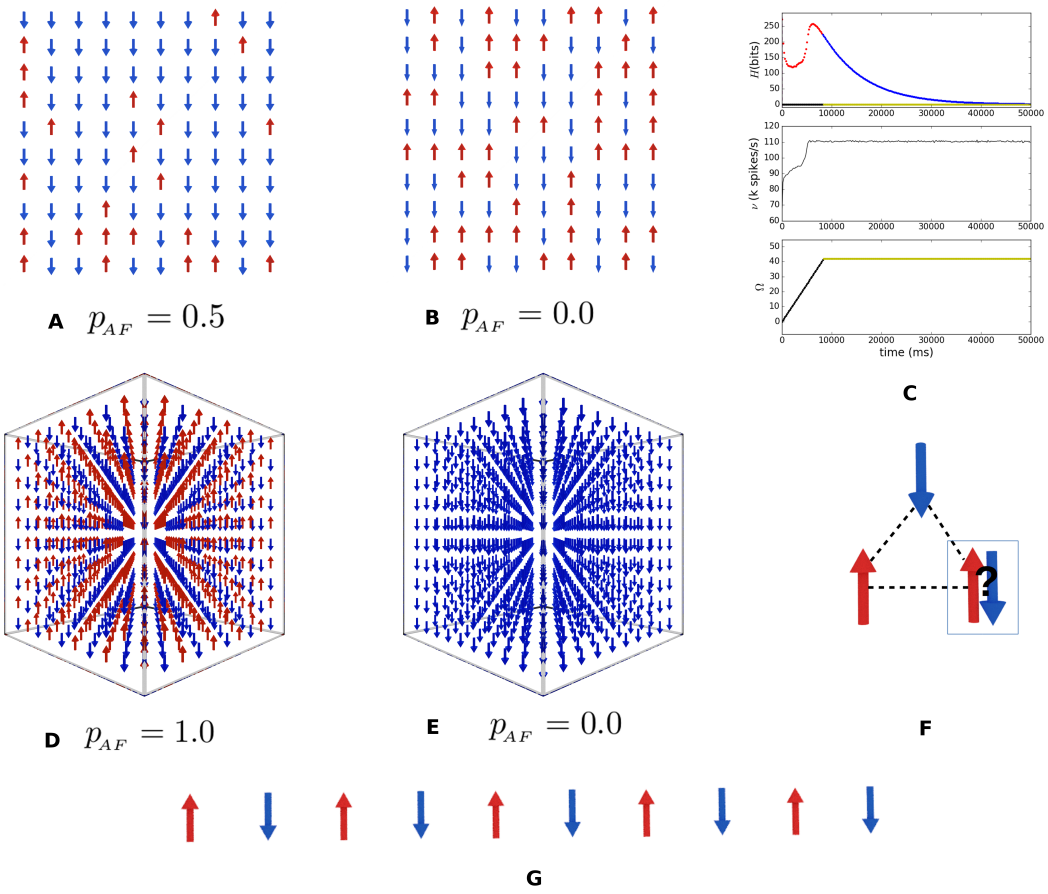


Figure 2.5: Spiking neural network simulation of Ising spin systems. (A) and (B) show two 2-dimensional spin glass quenched states obtained with interaction probabilities $p_{AF} = 0.5$ and $p_{AF} = 0.1$. The results for the 3-dimensional lattices for CSPs of 1000 spins with a ferromagnetic and an antiferromagnetic coupling constant are shown in (E) and (D) respectively. In (C) are plotted the temporal dependence of the network entropy, firing rate ν and states count Ω during the stochastic search for the system in (D). (F) illustrates the origin of frustrated interactions in spin glasses. (G) depicts the result of the 1-dimensional chain. The parameters for the SNNs used are shown in table 1.

system is then given by the Hamiltonian operator:

$$\hat{\mathcal{H}} = -\sum_{i,j} \mathcal{J}_{i,j} \vec{S}_i \vec{S}_j - h \sum_i S_i \tag{2.6}$$

where h is an external magnetic field which tends to align the spins in a preferential orientation [Bar82]. In this form each $\mathcal{J}_{i,j}$ defines a constraint $C_{i,j}$ between the values $D = \{+1, -1\}$ taken by the variables $\vec{S}_i$ and $\vec{S}_j$. It is easy to see that the more constraints are satisfied, the lower becomes the value of H in equation 2.6. This simple model allows the study of phase transitions between disordered configurations at high temperature and ordered ones at low temperature. For ferromagnetic $\mathcal{J}_{i,j} > 0$ and antiferromagnetic $\mathcal{J}_{i,j} < 0$ interactions the configurations are similar to those in figures 2.5D and 2.5E for 3D lattices, which correspond to the stable states of our SNN solver when the Ising models for $\mathcal{J}_{i,j} > 0$ and $\mathcal{J}_{i,j} < 0$ are mapped to an SNN using algorithm 2.1 and a 3D grid of 1000 spins. Figure 2.5G shows the result for a 1D antiferromagnetic spin chain. It is interesting to note that the statistical mechanics of spin systems have been extensively used to understand the firing dynamics of SNNs, presenting a striking correspondence between their behaviour even in complex regimes. Our framework allows the inverse problem of mapping the SNN dynamics to spin interactions. This equivalence between dynamical systems and algorithms allows emulation between equivalent dynamical systems. However, the network parameters should be adequately chosen to keep the computation valid.

If instead of fixing $\mathcal{J}_{i,j}$ to some value U for all spin pairs $\{(i, j)\}$ one allows it to take random values from $\{U, -U\}$ with probabilities p_{AF} and p_{FM}, it will be found that certain interactions would be frustrated (unsatisfiable constraints). Figure 2.5F illustrates the frustration with three antiferromagnetic interacting spins in a way that any choice of orientation for the third spin will conflict with one or the other. This extension of the Ising model when the grid of interactions is a random mixture of AF and FM interactions was described by [EA75]. The model is the representation of the spin-glass systems found in nature, these are crystals with low concentrations of magnetic impurities which, due to the frustrated interactions, are quenched into a frozen random configuration when the temperature is lowered (at room or high temperatures the magnetic moments of a material are constantly and randomly precessing around their average orientation). The statistical analysis of those systems was fundamental for the evolution of artificial neural networks and machine learning. Furthermore, the optimisation problem of finding the minimum energy configuration of a spin glass has

been shown to be NP-complete by [Bar82]. The quenching of the grid happens when it gets trapped in a local minimum of the state space of all possible configurations. In figures 2.5A and 2.5B we show a quenched state found by our SNN with $p_{AF} = 0.5$ and $p_{AF} = 0.1$ respectively. A spin glass in nature will often be trapped in local minima and will need specific temperature variations to approach a lower energy state; our SNNs replicate this behaviour and allow for the study of thermal processes, controlling the time variation and intensity of the excitatory and inhibitory stimulations. If the underlying stochastic process of such stimulations is a good representative of heat in solids, they will correspond to increase and decrease of temperature respectively, allowing, for example, the implementation of simulated annealing optimisation. Figure 2.5C shows the time evolution of the entropy, firing rate and states count for the antiferromagnetic 3D lattice of figure 2.5D, similar plots but converging to unsatisfying states are found for the spin glasses in figures 2.5A and 2.5B. In the case of the ferromagnetic lattice in 2.5E with very low noise, the network immediately converges to a solution, if the noise is high, however, it is necessary to stimulate the network several times to have a perfect ordering. This is because more noise implies more energy to violate constraints, even in nature, magnetic ordering is lost at high temperatures.

2.7 Discussion

The examples of the last section show the basic features of the stochastic search and the use of the entropy, firing rate and the number of states to track the behaviour of the network. In order to evaluate the performance of the search, we have performed a series of runs for each simulation until the network has been successful 100 times. The histograms of the corresponding convergence times for each example are shown in figure 2.6, also displaying the mean μ, standard deviation σ, skewness γ_1, success ratio ξ (defined as the number of times the simulation converged to satisfaction over the total number of runs) and the best convergence time t_{min} of each underlying experiment. The dimensions of the SNNs and simulation parameters for the three CSPs shown here are summarised respectively in tables 2.1 and 2.2.

The hard Sudoku puzzle of figure 2.4 was previously solved using spiking [HJM13] and rate-based [MMI15b] neural networks with mean solving times of $29\,s$ and $153\,s$ respectively. The solver presented here reduces the mean solving time for this puzzle to $6.36\,s$ implying a considerable improvement in performance for Sudoku neural solvers. The same network parameters were used to solve the three Sudoku puzzles in order to

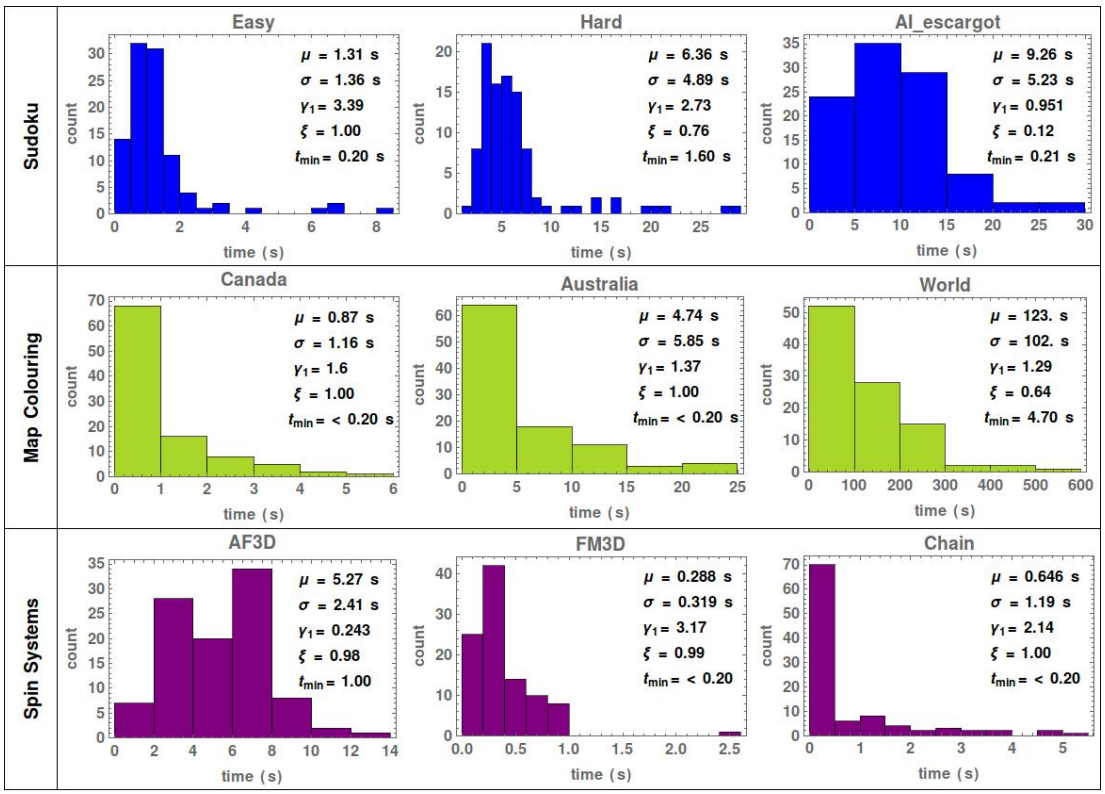


Figure 2.6: Histograms of the convergence time to a solution for the Sudoku, map colouring and spin system problems of figures 2.2, 2.4 and 2.5. For each histogram data from 100 simulations were used. The mean μ, standard deviation σ, skewness γ_1, success ratio ξ and the best convergence time t_{min} are indicated for each problem. The success ratio is defined as the number of times the simulation converged to satisfaction over the total number of simulations.

show the relationship between the stochastic search and the puzzle difficulty. Clearly, the average time for convergence increases with difficulty, but more significant is the strong decrease of the success ratio. Thus, to avoid overfitting, a trade-off between exploratory and greedy behaviour needs to be found for the problem at hand. The state of the art Sudoku solvers (see for example [Don12, Nor09]) are able to solve puzzles in tens to hundreds of microseconds. Such solvers use backtracking together with deductive methods specific for Sudoku. Consequently, they are not general-purpose as the one presented here. It is precisely the specificity what provides their speed-up.

The solution to the map of the territories of Canada, as defined in figure 2.2, was presented by D-Wave systems to demonstrate the applicability of their quantum computer. To find the solution, they executed a quantum machine instruction which can return 10000 samples/s from which $\approx 25\%$ solved the problem [Hea13]. This means an effective time to solution of $0.4ms$. The power consumption of the machine is $25kW$, and it operates at a temperature of $0.015K$. For this same map, our solver uses three SpiNNaker chips each one consuming at most 1W of power, and it finds the solution with a mean of 0.87 s. Additionally, classical techniques like simulated annealing [CHDW87], genetic algorithms [GLH93] and tabu search [DH99] as well as the more elaborated state-of-the-art algorithms [GWTH10, CS$^+$02, LH10, HPZ08, BZ08, GH06, CS$^+$02, FLS01, DH99, GLH93, CHDW87, TC11], solve colouring map problems in time scales ranging from tens of seconds to tens of thousands of seconds and conventionally have a success ratio below 1 for the allocated time. As seen in 2.6, this is the same order of magnitude for the time that our SNNs needed to solve the colouring map problems of figure 2.2.

It is then verified that the solutions found by the SNNs in SpiNNaker are on the order of magnitude of interest for biological systems, these require solutions in hundreds of *ms*, which is the natural timescale of consciousness [TBMK16]. This is also the timescale of performance of classical general-purpose solvers [GAD$^+$13]. In contrast, Problem-specific solvers can find solutions in a few μs through heuristics specific to the problem, for example, mimicking how humans solve Sudoku. If the problem is not solvable by the presumed strategies the solver will not perform well. To leverage neuromorphics to main-trend computing it is then desirable to find solutions in the order of μs, for this one could resource to accelerated hardware, e.g. BrainScales [SBG$^+$10] which runs 10000 times faster than real-time (biological real-time resolution is *ms*). However, these systems are still limited by the number of neurons and synapses they can handle. Better performance is also expected from the second generation of SpiNNaker,

which is currently under development (see section 1.2). We turn instead in the next chapter to use Loihi, a state-of-the-art neuromorphic chip developed by Intel which keeps the scalability of SpiNNaker while approaching the speed of BrainScales, with it we improve the performance and efficiency of our solver to some hundreds of μs. It is also important to highlight that the NP feature of an algorithm refers to its increasing complexity with the size of the problem and that the problems presented here correspond to instances of expressly modest sizes. Nevertheless, the number of variables for most problems in robotics and perception have an order of magnitude comparable to that of these CSPs.

The main advantage of stochastic search algorithms is that they are general-purpose, able to find satisfactory solutions without needing much detail about the specific problem at hand. Moreover, the exploration of solutions to constraint satisfaction situations never seen before is the typical way in which organisms explore the environment and acquire knowledge about it. To build the solvers of the previous section, we have used only the number of variables, domain size and constraints list, nevertheless the network showed good performance. Thus if a system of SNNs is able to collect this kind of information from its environment, it will easily take beneficial decisions.

Future work involves the extension of the framework to solve optimisation problems where the constraints are defined by inequalities (e.g. to solve the travelling salesman problem or to find the minimum energy configuration of a spin glass), or other more general non-linear constraints. A concern with such class of problems is that the network needs to be able to recognise the best option among all the configurations that satisfy the constraints, identify a nonzero energy minimum or explicitly model the cost function. In such cases, more advanced techniques or non-stochastic strategies can be necessary to achieve convergence. In this regard, techniques from nonlinear programming could guide the improvement of SNN solvers in decision making under more complex constraints. In chapter 4, a step forward in this direction is taken, the cost function is explicitly evaluated by the SNN, which is also able to recognize its minimum, thus eliminating the need for convergence.

2.8 Conclusions

In summary, we have presented a neuromorphic implementation of spiking neural networks stimulated with Poisson spike sources which solve constraint satisfaction problems. The network dynamics implements a stochastic search over the problem's space

of states which, with an adequate choice of parameters, is able to converge to a stable configuration (or set of configurations) that satisfy all the constraints. Satisfactory performance was found, and further research is needed for CSPs defined by more complex constraints. Furthermore, we presented a software framework to explore new strategies for stochastic searches with SNNs. The code of the framework and examples presented here is made available at `https://github.com/GAFonsecaGuerra/SpiNNakerCSPs`.

Table 2.1: Network sizes of the SNN solvers of the CMP, Sudoku and Spin Systems

Network Parameters				
CSP	number of neurons	number of synapses	populations (number of variables)	sub-populations (domain size)
World CMP	212400	14422300	193	4
Australia CMP	450	22920	7	3
Sudoku Easy	36675	86154125	81	9
Sudoku Hard	36675	86154125	81	9
AI Escargot	36675	86153250	81	9
AF Ring	1050	975500	10	2
Spin 2D Lattices	10050	2160000	100	2
Spin AF 3D Lattices	100050	31050000	1000	2
Spin FM 3D Lattices	100050	31050000	1000	2

Table 2.2: Simulation parameters for the SNN solvers of the CMP, Sudoku and Spin Systems

Simulation Parameters				
CSP	Noise Populations stimulation (depression)	Internal Inhibition Weights	Constraints Strength Weights	External current
World CMP	10	[-0.08, 0.0]	[-0.08, 0.0]	0.3
Australia CMP	1(1)	[-1.2, -1.5]	[1.2, 1.4]	0.2
Canada CMP	1(1)	[-1.2, -1.5]	[1.2, 1.4]	0.17
Sudoku Easy	1(0)	[-0.08, 0.0]	[-0.08, 0.0]	0.3
Sudoku Hard	1(0)	[-0.08, 0.0]	[-0.08, 0.0]	0.3
AI Escargot	1(0)	[-0.03, -0.02]	[-0.03, -0.02]	0.3
AF Ring	1(0)	[-0.2, 0.0]	[-0.2, -0.0]	0.0
Spin 2D Lattices	1(1)	[-0.2, 0.0]	[-0.2, -0.0]	0.0
Spin AF 3D Lattice	1(0)	[-0.2, 0.0]	[-0.2, -0.0]	0.0
Spin FM 3D Lattice	1(0)	[-0.2, 0.0]	[-0.2, -0.0]	0.0

Chapter 3

The Intel Loihi Neuromorphic Chip

3.1 Introduction

In this chapter, we review the hardware and software architectures of Intel's neuromorphic research chip Loihi (pronounced low-ee-hee), which will be used in chapter 4 to improve on the results of the previous chapter. Intel named the chip after a volcano in Hawaii emerging from the sea, highlighting its role as an emerging and promising technology for cognitive computing. We have mentioned how the neuromorphic paradigm did not reach widespread commercialization despite its early formulation more than 30 years ago [Mea90]. This has been similar for SNNs, for which the primary limitations have been their costly implementations in conventional hardware based on the von-Neumann architecture, as well as the lack of principled algorithms which solve real-world problems. Despite this autumn, SNNs include biological features relevant for cognition and absent from the main trends in machine learning and AI, e.g. deep neural networks (DNNs).

AI still differs considerably from natural or general intelligence, its algorithms are power-hungry and demonstrate robust but slow learning, making it clear that an affordable domestic-sized real-time cognitive computer, unlike IBM's Watson, is not achievable with ANNs implemented in accelerator architectures for matrix computing, unless dependent on the cloud. SNNs, on the other hand, harness sparsity in both space and time with computational advantages not fully uncovered yet. Thus, to overcome the historical limitations of SNNs and neuromorphic hardware, Loihi has been developed for research, aiming to shed light on the real computational power of event-driven time-dependent neural networks, focusing primarily on mathematical rigour, principled models and rapid architecture iteration [DSL$^+$18]. The road-map

is set to understand and benchmark the capabilities of neuromorphics for commercial applications, improve current architectures and paradigms, as well as develop and test principled SNN algorithms [Dav19]. Loihi was announced in September 2017 and launched on the 2018 Neuro Inspired Computational Elements (NICE) workshop in Oregon US. Intel's neuromorphic endeavour is pushing the boundaries of the field and assessing its value for applications under the best available manufacturing constraints. Equally to previous neuro-inspired devices, the foreseen computational advantages are efficiency and speed, including the possibilities of real-time and accelerated computing. In the next chapter, we demonstrate gains in performance and energy efficiency for our CSP solver by using Loihi's novel feature set.

Software development in a prototype chip process should consider the unforeseen implementation of alternative solutions, as well as the uncertainty in the functionality of the work-flow. Nevertheless, we will show in the next chapter, how the flexibility offered by the chip's design, together with the feasibility of future iterations, makes the development in Loihi encouraging. We hope that the intense research will lead to better features as has been done with PCs and smartphones, accelerating the field to the next level of smart and adaptive computing, one that satisfies the goals of cognitive computational neuroscience, i.e. cognitive models that perform tasks [KD18]. The last section of this chapter will show the progress that Loihi has enabled in this direction.

3.2 Hardware Architecture

3.2.1 The Loihi Chip

Loihi is a digital manycore neuromorphic processor which in its current iteration contains 128 neurocores per chip, adding up to 131072 single-compartment spiking neurons, and allowing to store up to 130 million synapses through its various compression formats. The chip works with supply voltages in the range of 0.50 V to 1.25 V. It contains 2.07 billion transistors on its die size of 60 mm^2 and is fabricated in Intels FinFET 14*nm* CMOS process technology [DSL+18]. Besides the neurocores, the chip includes three embedded x86 Lakemont (LMT) CPUs. The integration of synchronous and asynchronous designs on the same SoC is alleviated by specifying the architecture through the Communicating Sequential Processes Language, which is then translated into Verilog RTL [LJL+18].

Loihi's architecture is fully asynchronous with the message-passing channels implemented as bundled-data (single-trail) asynchronous pipelines [LJL+18]. In bundled-data style, the handshaking is bundled with the data signals. However, in contrast with SpiNNaker's 3-of-6 return-to-zero, here the protocol is 2-phase (non-return-to-zero) in which the request and acknowledgement tokens are encoded as signal transitions, this protocol generally leads to faster circuits than those encoding the handshaking as Boolean levels (return-to-zero) [SF02]. The physical NoC is formed by an 8×16 mesh of neurocores grouped in 4-core close-contained tiles with one router per tile and no need for a global clock. The actual number of tiles per chip is somehow arbitrary and can easily be changed to build bigger or smaller chips. Counting both neurocores and LMTs the chip totals 33 MB of SRAM memory (16 MB of synaptic memory).

The chip is programmable in terms of network topology, and neural dynamics inside the constraints of its neuron model. The architecture has been optimised to exploit sparsity, using a pointer and value encoding with routing tables similar to SpiNNaker [FLP+13] and in contrast to crossbar architectures used in other chips, which may waste resources when non-dense adjacency matrices define the network. Loihi maintains functional determinism and introduces a new set of features when compared to previous neuromorphic designs: population-based hierarchical connectivity, dendritic compartments (multicompartment neurons), programmable learning rules through an on-chip learning engine and variable synaptic formats with 1 to 9 bits precision weights [DSL+18, LWC+18a]. In operation, each core receives input spikes from other cores, iterates sequentially over its neuron compartments, integrating input and updating their state variables according to the neuron model, finally, it generates output spikes to be routed by the NoC. Spikes are sent to the set of neurons in the mesh to which a postsynaptic connection exists. When the process is completed, a barrier synchronisation acknowledges advancement to the next timestep in which the process is repeated. The chip advances through time discretely, but the core load determines the length of the timesteps, each processing unit advances as fast as it can to complete its scheduled tasks, the timestep is finished when all units have done so.

Regarding synaptic density, when using 1-bit synapse format Loihi supports 2.1 million unique synapses per mm^2, surpassing TrueNorth by a factor of 3, which was regarded as the densest SNN chip. The effective synaptic density can be further improved by exploiting Loihis hierarchical connectivity. Notice, however, that using binary synapses and hierarchical connections impose certain constraints on the type of SNNs to be implemented. Concerning neuron density, Loihi's augmented feature set and

flexibility yield 2,184 neurons per mm^2 which is about half of TrueNorth's maximum neuron density. This is partly due to the richer feature set compared to TrueNorth, such as neural homeostasis, presynaptic and postsynaptic activity variables in support of on-chip synaptic plasticity Given the current status of neuromorphic hardware, a level of programmability and power efficiency which enables mobile and robot devices are its more essential features. Thus, the differences above betweem TrueNorth and Loihi, all under one order of magnitude, are marginal and we expect them to be improved in future iterations of both chips.

3.2.2 Neuron Model

The neurocores host a flexible number of compartments built over a current based (CUBA) leaky integrate and fire (LIF) spiking neuron model. Compartments can be grouped to form multicompartment neurons with an arbitrary binary-tree structure or act independently as single compartment LIF neurons. Each compartment is governed by two internal state variables u and v corresponding to their synaptic response current and membrane voltage, respectively. A compartment i can also have an intrinsic bias current b_i which models its intrinsic excitability and drives v_i. When v_i reaches the voltage threshold θ_i from below, the compartment emits a spike message and v_i is reset to its resting potential where it remains for certain refractory period. In practice, the spiking state constitutes a third state variable $s_i(t) = [v_i(t) > \theta_i]$. When $s = 1$, the compartment sends its spike to the set of destination neurons. Each postsynaptic neuron j then integrates the incoming spikes associating a synaptic weight w_{ij} to an event arriving from neuron i. Biologically, w_{ij} represents the charge transferred at the synaptic cleft by the incoming action potential. Such input charge elicits a postsynaptic response in j which is characterised by an exponential filter of the form:

$$\alpha_u(t) = \frac{1}{\tau_u} e^{\frac{-t}{\tau_u}} \Theta(t) \tag{3.1}$$

where $\Theta(t)$ is the Heaviside step function.

Hence, a spike train $\mathbf{s}_j(t) = \sum_k \delta(t - t_j^k)$ arriving at compartment i causes a sequence of filtered and weighted input currents on top of the compartment's intrinsic bias

[TLD17]:

$$u_i(t) = \sum_{j \neq i} w_{i,j}(\alpha_u * \mathbf{s}_j)(t) + b_i, \tag{3.2}$$

$$\dot{u_i} = -\frac{1}{\tau_{u_i}} u_i(t) + \sum_{j \neq i} w_{i,j} s_j(t) + b_i. \tag{3.3}$$

The current u_i is then integrated by the neuron into its compartment voltage through:

$$\dot{v_i} = -\frac{1}{\tau_{v_i}} v_i(t) + u_i(t) - \theta_i s_i(t), \tag{3.4}$$

where the dot denotes time derivative, and τ_v controls the cell membrane leakage. Notice that the last term in equation 3.4 accounts for the voltage reset when the compartment fires. This happens whenever $v_i(t) > \theta_i$ which correspond to the i neuron spike train $\mathbf{s}_i(t)$, thus the term $\theta_i \mathbf{s}_i(t)$ is non-zero only when neuron i spikes.

Given the digital nature of the chip, the continuous dynamics defined by equations 3.3 and 3.4 is approximated by difference equations with a common timestep Δt. As noticed in the previous section, this fixed-length timestep is algorithmic rather than physical. Because the chip is fully asynchronous, a synchronisation barrier is used throughout the entire system to coordinate the simulation. After every core finishes updating the computations for a given timestep, it hand-shakes a barrier synchronisation message with its neighbours, when all cores have gone through this process, an advance timestep message propagates so that the simulation advances. Hence, the physical time associated with Δt potentially varies from cycle to cycle in a timescale of microseconds. Such variability will not affect the SNN algorithms because all updates have to be finished as a prerequisite for the barrier synchronisation.

The discrete version of equations 3.3 and 3.4 is:

$$\frac{\Delta u_i}{\Delta t} = -\frac{1}{\tau_{u_i}} u_i(t) + b_i + \sum_{j \neq i} w_{i,j} s_j(t), \tag{3.5}$$

$$\frac{\Delta v_i}{\Delta t} = -\frac{1}{\tau_{v_i}} v_i(t) + u_i(t) - \theta_i s_i(t). \tag{3.6}$$

From which we obtain the updating equations:

$$u_i(t) = u_i(t-1) \cdot (1 - \frac{\Delta t}{\tau_{u_i}}) + \Delta t b_i + \sum_{j \neq i} w_{i,j} s_j(t) \Delta t, \tag{3.7}$$

$$v_i(t) = v_i(t-1) \cdot (1 - \frac{\Delta t}{\tau_{v_i}}) + u_i(t) \Delta t - \theta_i s_i(t) \Delta t, \tag{3.8}$$

In units of Δt:

$$u_i(t) = u_i(t-1) \cdot (1 - \delta_{u_i}) + \sum_j w_{ij} \cdot s_j(t) + b_i \tag{3.9}$$

$$v(t) = v(t-1) \cdot (1 - \delta_{v_i}) + u_i(t) \tag{3.10}$$

In equations 3.9 and 3.10 $\delta_{u_i,v_i} = 1/\tau_{u_i,v_i}$.

3.2.3 Multicompartment Join Operations

The ability to build multicompartment neurons is an essential feature for our self-validating CSP solver demonstrated in the next chapter. Without these, we would still have achieved improvement in speed and energy due to the hardware features of Loihi, but would not have a self-validating network which avoids probing u, v and s.

To build multicompartments, each neurocore uses an auxiliary memory stack. While the runtime software is advancing the computations corresponding to one timestep, any state variable from a compartment can be pushed into the stack and be used by the next logical compartment. This protocol is possible due to the sequential updating of compartments in the same neurocore. When allocating resources, the order in which the compiler maps the compartments in the hardware will reflect the direction of the flow of information in the multicompartment neuron. This usually corresponds to the order in which the compartments are created using the high-level API that programs the chip. Loihi enables a set of multicompartment operations which use the stack to propagate state variables across several compartments, practically building binary trees of compartments (multicompartmental neurons). Formally speaking, the functionality of these multicompartment trees reflects only partially the operation of biological neurons; however, they bring extra computational abilities to the space of SNN-based algorithms. Multicompartment operations tell a compartment whether and which information to

Table 3.1: Neurocore resource constraints for Loihi [DSL+18].

Magnitude	Maximum
Neurons per core	1024
Synaptic Fan-in state	128 KB
Core-to-core Fan-out edges	4096
Distribution lists associated by axon ID	4096

gather from and send to the stack for sharing throughout the neuron.

3.2.4 LFSR Random Number Generator

Loihi includes an in-hardware 8-bit linear-feedback shift register (LFSR) pseudo-random number generator to enable efficient stochastic computations, as well as the inclusion of noise in biologically relevant simulations. The LFSR unit returns an 8-bit uniformly distributed random number between 0 and 255. The number is then centred by substracting 2^7 and can also be shifted and scaled respectively by the software specified parameters noiseMantOffset and noiseExp:

$$\eta = (LFSR() - 2^7 + noiseMantAtCompartment * 2^6) * 2^{noiseExpAtCompartment-7} \quad (3.11)$$

These uniformly distributed random numbers allow randomisation of either the compartment voltage, compartment current or the refractory delay. The LFSR seed can be modified to define the statistical average behaviour of the simulations. Given the fact that the chip dynamics is deterministic, several simulations run with the same seed for the LFSR will yield the same results, this is advantageous when designing and testing algorithms based on stochastic SNNs.

3.2.5 Core Design

Cognitive tasks in nature are performed with reliability through high spatial redundancy, which is partly due to internal and environmental noisy and unpredictable conditions. The neurocore design for Loihi tackles the intrinsic unpredictability with determinism, even under stochastic computations by implementing a barrier synchronisation which forces a common sense of discrete time across the SNN and keeps the dynamics well defined. Moreover, by using multiplexing and linear pipelining it shrinks circuit area,

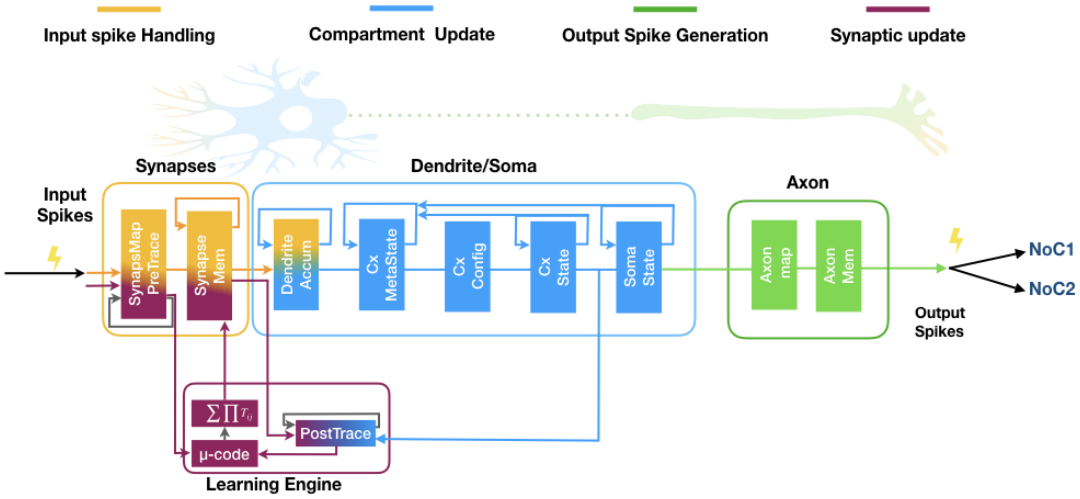


Figure 3.1: Internal structure of a neurocore in Loihi.

converting spatial redundancy in temporal unfolding. Thus, compensating for the disadvantages of transistors when compared to the molecular machinery developed by nature, which despite highly granular and parallel, executes a slow dynamics compared to integrated circuits operating at hundreds of GHz.

The neurocore distributes a variable number of compartments across ten architectural memories arranged in tree-like structures, figure 3.1. These memories form input synapse, dendrite and axon output blocks, as well as a learning engine on top communicating with the synapse and dendrite segments. Four operating modes can be defined: input spike handling, neuron compartment updates, output spike generation, and synaptic updates. The compartments in a core share the fan in and fan out connections as well as redundant parameters. These, in turn, are used to update the parameters unique to every compartment at every algorithmic timestep. This approach considerably improves efficiency. Some memory blocks can be used for more than one functionality allowing for the number of neurons per core to be variable and flexibly support distinct features without wasting resources.

Memory blocks operate largely independently of each other allowing for a diversity of frequencies with each mode executing as fast as it can, in sequential or parallel processing and with minimal synchronisation. Wherever events can be expanded into smaller events, e.g. synaptic and learning pathways, the hardware implements fine-grained parallelism.

The total SRAM memory in the neurocore adds up 2Mb (including error-correcting

code), which supports a maximum of 1024 neurons per core (2,184 neurons per mm^2). One particularity of the chip is its support for read-modify-write memory access (shown as loops around memories in figure 3.1), instead of just read-write access. This is an essential feature which enables the above design.

In operation, each incoming spike encodes the index of its sending axon which points to a pool of synaptic entries (memory limited up to 128Kb) encoding a tuple of the form (weight, delay, fanout neuron, tag) [LJL+18]. A ring buffer accumulates the weight for the future updates of each compartment and the compartment consumes the accumulated weight for the current timestep. Produced spikes follow to the end of the pipeline to a list of destination cores and axons. Such routing tables are also constraint by memory size, table 3.1.

3.2.5.1 Configurable Delays

Configuration of both synaptic, refractory and axon delays is possible. Axonal delay is shared and refers to the delay of spikes fired in a core before they get distributed to postsynaptic neurons. This is analogous to the time it takes for a spike to travel from the axon hillock to the axon terminals.

The refractory delay refers to a period during which the neuron would not update its compartment voltage. When using box PSPs, the refractory delay may coincide with the box duration to abstract the duration of the effect of a spike in the network and onto itself.

Discrete (not shared) synaptic delay. After the axonal delay, spikes are distributed through the NoC, on arrival to the destination compartment the spikes can be further delayed by a set of dendritic accumulators. Similar to what will take a spike to transverse a biological dendrite to reach the soma. The accumulators temporally buffer the spikes until these get integrated by the compartment response current by consuming the accumulated synaptic weights w_{ij}. Increasing the number of accumulators result in fewer neurons per core, so we try to avoid such modification. In fact, as one of the accumulators is associated with the current timestep, the maximum delay is seven timesteps. This delay type, however, also specifies the duration of the box PSP, which is used for the CSP Loihi solver. Sometimes though, the equilibrium state of the network encodes the problem solution sparsely in time, requiring averaging over long time windows for decoding. This is not a problem if offline validation based on post-processing is used, as in the previous chapter. It will be seen, however, that for online CSP validation the PSP length imposes a constraint on the detection of solutions

encoded sparsely in time. To avoid using more dendritic accumulators which would increase the number of cores needed for a given network, we present an additional neural layer which can buffer the network state during the windows of sparsity where the network state is under-determined.

3.2.5.2 Activity Phases

Through its dynamics, a compartment transitions between several phases. The user can hard-code these phases at either the beginning of the simulation or during runtime through the x86 CPUs. This latter can also read the phase of a compartment and send it to the host for post-processing. This is particularly useful, as we will see in the next chapter when the network state has to be reconstructed with some delay due to the time it takes for information to propagate across the hardware stack. The possible phases for a compartment are:

- the compartment dynamics does not get updated.
- the compartment is inactive after reaching a lower limit of the compartment voltage, returns to active after new excitatory input.
- regular updating of dynamics
- fired last timestep.
- the neuron is in the refractory period
- inhibited only for the duration of the refractory delay.

3.2.6 Asynchronous Communication

Because Loihi implements a mesh-level barrier synchronisation by handshaking tokens between cores, there is no mesh-wide idle time, as soon as all cores finish their operation, the mesh advances to the next algorithmic timestep. The NoC virtually supports multicasting by iterating over the routing tables of each core and physically using two independent unicast networks (to prevent deadlock) targeting destination cores iteratively and sending one spike per core.

In contrast with SpiNNaker's toroidal mesh, Loihi chips connect in four planar directions through off-chip interfaces, extending the on-chip multicast to the chip-to-chip communication allowing synapses between neurons in any two parts of the machine.

As it is typical for k-ary n-cube interconnects, events are distributed across chips hierarchically according to a dimension-order algorithm. Chip-to-chip communication also follows barrier synchronisation to advance in time.

Besides spike messages for SNN computations, and barrier messages for event-driven synchronisation, the NoC transmits write, read request and read response messages for both core management and communication between the embedded LMT CPUs. Off-chip communication scales up to 16 384 chips in each of which up to 4096 cores can be addressed. Because any of the packetized messages may be sourced externally by the host or on-chip by the LMTs, it is possible to define communication channels which interface with various latencies with an SNN running on the machine. We use such a protocol to identify solution times to the CSP networks and make Loihi report the respective solutions to the superhost.

3.3 Loihi-Based Systems

Loihi is the base chip for several hardware systems, listed in table 3.2.

3.3.1 Kapoho Bay

Kapoho Bay is the USB stick form factor of Loihi systems. Besides the USB connectivity to host, this modular system comes with GPIO pins, I2C and a DVS AER Interface, allowing connectivity to external devices such as DVS cameras. Our Loihi CSP solver explained in detail on the next chapter fits particularly well Kapoho Bay. Despite counting with only two chips, this device sums up to 256 cores with 262,144 compartments and 256 million synapses. Making it suitable for embedded decision-making in robotics and autonomous vehicles. In particular, our on-chip validation solver is made of 3-multicompartment neurons, one-hot-enforcement neurons and a summation neuron, the two latter needing a core each due to their different compartment prototype configurations. Thus, Kapoho Bay gives support for up to 86698 of our 3-multi-compartment neurons (260096 compartments) which can be distributed across CSP variables and their domains. When used with external devices, these can input cues to the CSP solver, which then sends a decision to, for example, an actuator for action execution. This process would endow a robot with active inference capabilities [FFR+16].

Our solver, however, scales to any of the other systems shown in table 3.2, as long

as the constraint mapping does not exceed the maximum hardware synaptic density.

3.3.2 Nahuku Board

Nahuku is the machine we used for development during this thesis. It consists of a 32 chip circuit board augmented by an Intel Arria 10 SoC Development Kit. The board allows the implementation of up to 4 million compartments with up to 4 billion synapses.

The Intel Arria includes a hard processor system with integrated ARM Cortex-A9 MPCore processor and an FPGA (Arria 10 SoC, 10AS066N3F40E2SG, 1517-pin FBGA). We communicate with the Arria board via ethernet through an external superhost server which hosts our code and the NxSDK 3.4. Both the ARM and FPGA portions of the Intel Arria 10 SoC allow increased flexibility for the interaction of Loihi with peripheral devices.

Users connect via Intel Xeon-based servers which use SLURM to handle job scheduling and act as superhost. Xeon CPUs offer high core count and allow increased RAM and cache compared to CPUs for regular workstations. SLURM is a management and job scheduling system which scales from small to large Linux computing clusters. It allocates jobs assigning hardware resources to users, manages the starting, execution and monitoring of several jobs in parallel and arbiters a queue of jobs to be executed as hardware becomes available [YJG03].

3.3.3 Pohoiki Springs

24 Nahuku boards build Pohoiki Springs. This system is under development and is designed to support over 100 million compartments and around 100 billion synapses distributed across 768 chips. This sums up to a representability of more than 30 million CSP variables (3-multi-compartment neurons). These values are, of course, limited by the constraints density of the CSP. Aligned with the design choices for this massive system, the Loihi CSP solver can also be used seamlessly to execute several equivalent CSP networks in parallel, initialising them with different parameterisations to speed-up the stochastic search. It is also possible to concurrently run several CSPs for a single experimental design (being these interacting or not).

Table 3.2: Hardware systems based on the Loihi Chip

System	Kapoho Bay	Wolf Mountain	Nahuku	Pohoiki Beach	Pohoiki Springs
Type	USB Form Factor	Board	Board	Multiboard Rack	Multiboard Rack
Max Number of Neurons	262144	524288	4194304	8388608	100663296
Max Number of Synapses	26×10^7	52×10^7	$4,16 \times 10^9$	$8,32 \times 10^9$	$9,984 \times 10^{10}$
Number of Chips	2	4	32	64	768

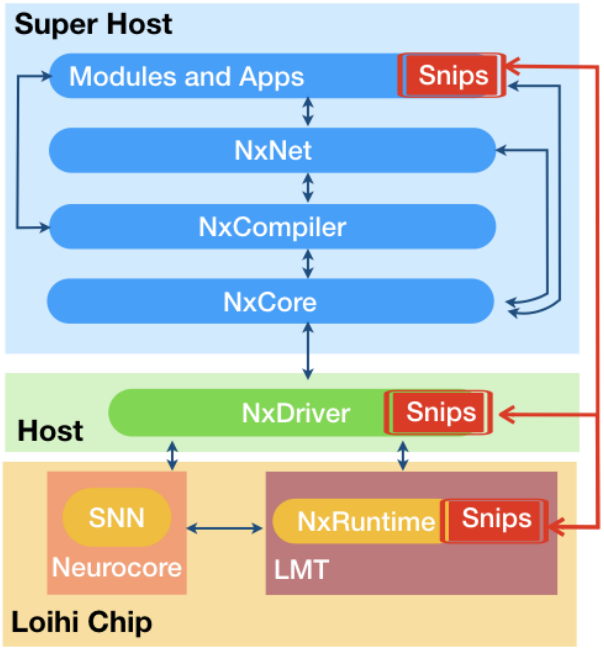


Figure 3.2: Hardware systems based on the Loihi Chip

3.4 Software Stack

Programming Loihi is done through a software development kit (SDK) named NxSDK, figure 3.2. A hierarchical software facilitates the user to tradeoff between execution speed, model complexity and programmability across the heterogeneous hardware stack. Under NxSDK a neuromorphic algorithm is composed of neural processes, sequential neural interfacing processes (SNIPs) and communication channels [LWC+18a].

Neural processes refer to the actual SNN, the definition of its topology, dynamics and learning rules, these run on the neurocores. At the object-oriented level, SNNs are determined through networks, neurons, terminals

and connections.

SNIPs refer to non-neural user-defined code to be executed by either the embedded LMTs or the external CPUs (FPGAs, host, superhost) SNIPs interact with the SNN at runtime actively (changing parameters or connectivity) or passively (reading network activity).

Communication channels allow information transfer across the different layers of hardware through packetized messages. In general, SNIPs communicate with the SNN via spikes.

The NxSDK additionally enables the probing of state variables, as well as a set of execution probes to measure energy and execution time assisting the benchmarking of the algorithms.

The software architecture is composed of several layers (figure 3.2). The highest level API NxNet runs on a super-host layer and allows the description of the SNN in terms of neural populations and their properties similar to PyNN [Dav08], the language we used to build our SpiNNaker solver. The users are expected to develop modules based on NxNet primarily. The high-level network specification is compiled by NxCompiler, which translates the NxNet description to the register level [LWC$^+$18b]. NxCore is a low-level API which allows the network specification in terms of registers and low-level functions. NxCore is generally intended for internal development rather than front-end use. However, to develop our CSP Solver we needed to make use of both NxNet, SNIPs and NxCore. The NxDriver orchestrates the runtime software NxRuntime from the host, as well as the LMT SNIPs. In the case of Nahuku, the NxDriver runs on the ARM processors of the Intel Arria 10 SoC, which communicates to the superhost via Ethernet.

3.5 SNN Mapping

The allocation of the SNN into the Loihi systems, carried out by the compiler, maps neurons to logical cores according to the hardware constraints. Notice that table 3.1 correspond to the maximum values; however, the network specification may impose further restrictions. If for example, 16 dendritic accumulators are used instead of 8 the maximum number of neurons per core will decrease from 1024 to 512. The compiler defines synaptic fan-in states by (weight, delay, tag) tuples and maps them into synaptic memory. Fan-out edges are represented by core-to-core links to each of which an axon ID is assigned to identify a unique destination core. The axon ID is then used by the

hardware as a pointer into a list of tuples of the form (j, weight, delay, tag) stored in the synaptic memory of the destination core.

3.6 Applications Development

Intel's commitment to neuromorphics has already benefited the field of neuromorphics with a few compelling demonstrations of the relative advantage of neuromorphic hardware on some problems over equivalent implementations in standard CPUs and GPUs.

Squares linear regression models with an L1 penalty on the regression coefficients, also known as Least Absolute Selection and Shrinkage Operator (LASSO) optimisation, has been solved in Loihi using the Spiking Locally Competitive Algorithm (SLCA) [LWC+18a]. The implementation demonstrates a 10^3 improvement in energy-delay product compared to state-of-the-art solvers implemented on conventional CPUs (1.67-GHz Atom CPU) running state-of-the-art algorithms (LARS and FISTA) while producing solutions within 1% of the optimal solution. This problem consists in optimising the l_1 distance between an input vector, and a set of feature vectors, such that the linear combination of feature vectors maximally represents the input vector. The SNN encodes the coefficients to be optimised through the bias current of a set of neurons. The SLCA appeals for applications where speed is more important than precision. The solution, in particular, harnesses the one-to-many characteristic of spike communication compared to the all-to-all nature of matrix-based algorithms such as FISTA, as well as the spike ordering by the explicit inclusion of time in SNNs.

Applied Brain Research's Nengo Deep Learning toolkit was used to implement a two-layer neural network spotter for speech recognition in Loihi [BCHE19]. Its performance and energy cost per inference was benchmarked against a CPU (23.2 $\times$), a GPU (109.1 $\times$), an Nvidias Jetson TX1 (20.5 $\times$), and the Movidius Neural Compute Stick (5.3 $\times$). This latter corresponds to a deep learning accelerator also developed by Intel. Still, Loihi demonstrated advantages in energy efficiency while maintaining the same level of recognition performance as the other platforms, a trend that improves with increasingly large networks.

Unidimensional simultaneous localisation and mapping (SLAM), in which a robot should figure out its pose and create a map of its surrounding environment, was demonstrated using Loihi [TSM19]. Multicompartmental and plastic SNNs were used to interpret dynamic sensory information, both visual and olfactory, and perform inference

on SLAM while the robot was spinning. Remarkably Loihi achieved comparable performance to a standard approach (the GMapping algorithm) while using only 1% of power against an Intel i7-4850HQ CPU.

Pre-silicon FPGA emulation of the chip demonstrated [DSL+18]:

- 10-neuron single-layer classifier with STDP-based supervised learning trained with local-intensity-change-based temporally spike-coded image samples. Loihi achieves 96% accuracy on the MNIST dataset.
- solved the problem of finding the shortest path on a weighted graph encoding nodes and links as neurons and plastic synapses.
- non-markovian one-dimensional decision making with reinforcement learning.

Loihi has already demonstrated low-power real-time object reecognition [May18], as well as an adaptive robotic arm controlling system [Eli18]. Finally, the next chapter will show how our own Loihi CSP Solver has achieved speed-up and improvements in energy efficiency with respect to our SpiNNaker implementation [FGF17].

Chapter 4

Loihi Solver for Constraint Satisfaction Problems with On-Chip validation

This chapter demonstrates a self-verifying multicompartment neuronal architecture for the efficient solution of constraint satisfaction problems (CSPs) on Intel's Loihi chip. In chapter 2, we already demonstrated the neuromorphic implementation of a scalable SNN-based solver for CSPs whose performance is on the same order of magnitude as that of the complete classical solvers for generic CSPs. To the author's knowledge that was the first neuromorphic implementation of problems of this size, previous and following implementations have been limited to small CSPs, like the 4×4 Sudoku puzzles [BIP16, Kug18]. Thus, the sPyNNCSP API demonstrates the applicability of the theoretical framework for stochastic computations with SNNs proposed by [JHM16, HJM13] who implemented their networks in conventional CPUs.

With the proof-of-concept, the next step is to optimize the solver to determine if it can be competitive enough to contribute to the widespread use of neuromorphic hardware. We can only achieve competitiveness if we demonstrate evident improvements in energy efficiency or execution time. There is enough room for optimising our sPyNNCSP in both hardware and software. In the hardware side, SpiNNaker's flexibility and ease to programming come at the expense of generous computational resources providing, for example, the ability to probe all state variables across the machine or simulate arbitrary dynamical systems (subject to fixed-point precision and memory restrictions). By harnessing off-the-shelf ARM processors, designed for general-purpose computing with fixed-point arithmetic, SpiNNaker's NoC architecture

includes some overhead local to the processors. Thus, although optimising sPyNNCSP is doable, in the interest of time, we shift to the most advanced neuromorphic chip available, Intel's Loihi. In doing so, we target performance comparable to, or better than, that of state-of-the-art solvers. It is important to highlight, in this chapter, we are not interested in comparison between neuromorphic platforms but rather on what these have to offer against conventional CPUs. A thorough benchmarking of SpiNNaker and Loihi chips would require performing the best feasible optimisation for each system, which is out of the scope of this thesis. As seen in chapter 3, Intel fabricated Loihi in its 14 *nm* technology (compare with ARM 968's feature size of 130 nm) with a fully asynchronous non-Von-Neumann architecture implementing digital spiking neurons. For our solver, its features give an estimated 15 times less power consumption, as well as 2-4 orders of magnitude improvements in execution time, depending on workload. While keeping a massive number of neurons per chip, contrary to accelerated analogue hardware, like BrainScales or DYNAPSE [SBG$^+$10, MQSI18], for which synapse and neuron counts are very low. In this regime, we aimed to produce a solver that could eventually compete at an industrial level. In this chapter, we find that it is possible to speed-up our neuromorphic CSP solver up to four orders of magnitude. The order of magnitude of the number of timesteps we need to find solutions in this chapter ranges between 10^2 and 10^3, while each timestep in Loihi takes at most few tens of μs. This means that our solver in this chapter finds a solution to modest size problems in $10^{-4} - 10^{-3} s$ (hundreds of μs to a few ms). State of the art classical solvers, with sophisticated solving strategies, solve problems of the size of those in this chapter in $10^{-2} - 10^{-1} s$ (see figures 2-4 and table 11 in [GAD$^+$13] generated with an Intel Core2 Quad CPU 3.00 GHz with 8 GB of RAM and a 64-bit operating system). Given the fact that even further optimisations are possible in Loihi, we firmly believe the solver could compete with conventional hardware and other novel post-von-Neumann architectures.

The most significant contribution of this thesis is presented in this chapter, consisting on the design and implementation of an on-chip validation neural architecture which endows the spiking neural network with the ability to measure its cost function in an online fashion, i.e., on-chip. Hence, the network can detect and report when it finds a solution to the encoded CSP. Counting with the solution time acknowledged at runtime, we implemented a set of SNIPs (see sequential neural interface processes in section 3.4), for extracting the solution as soon as the host gets notified. Our self-verifying architecture means that there is no need to probe the network at each time step, neither extract such information from the chip nor perform postprocessing. All we probe is

the firing of a single neuron, the reporter, whose spiking triggers a SNIP for solution readout to be run during the same timestep the reporter spikes. One of the Lakemont (LMT) embedded CPUs notifies the host the solving time. The host, in turn, requests the neurocores to provide their state variables corresponding to the network state which satisfied all the constraints. Finally, the host decodes the solution by applying a simple conditional over the state variables and decides whether the network continues exploring to find new solutions or stops the simulation altogether. Further versions of the solver can also include modifications to the CSP structure after solution retrieval. This turns the solver into a solver for dynamic CSPs. All in all, our results not only represent energy savings and speed up, but also the foundation for the implementation of sophisticated CSP solving strategies orchestrated by the embedded LMTs or other SNN motifs. The user-friendly API enables the use of the solver as a module in the creation of complex control systems. The solver presented here will be part of the next release of the NxSDK and is part of the endeavour to benchmarking Loihi's competitiveness for mainstream computing.

4.1 Network Energy Function

The target functionality for our Loihi CSP Solver (CSPNxNet) does not depend on convergence as it was the case in chapter 2. The reason is we will be able to spot solutions online, i.e. at runtime. Not requiring convergence means the network dynamics can stay highly noisy so that high-quality samples can be drawn from the state space of the CSP as the network evolves in time. A solution per se is not better than other for solvable problems because both should satisfy all constraints, nevertheless, external constraints, preferences or costs not included in the problem definition can make one solution more adequate than other (recall that checking a solution is trivial compared to finding it for NP problems). It is also conceivable to have applications, like configuring non-player characters in video games [CBBA14], in which several satisfying valuations, or even those with only a certain level of satisfiability, are desirable. This latter is essential for unsolvable problems. In a spin glass, for example, the existence of geometrical frustration means the energy function cannot reach 0. In this chapter, however, we only explore complete solutions to solvable problems.

Instead of measuring the network entropy as in chapter 2, we begin here by defining an energy function E which tracks the progress of the constraint satisfaction stochastic spiking neural network (CSSSNN or CSN) towards satisfying configurations. The

square adjacency matrix $\mathbf{W} \in \{1,0\}^{\mathcal{N}\times\mathcal{N}}$, entirely defines the topology of the principal population from the CSN, $\mathbf{W}$ encodes the existence of synapses between any pair of neurons from the $\mathcal{N}$ neurons composing the network encoding the CSP. The dynamic state of the network is defined, similarly to [HJM13], by a vector $\mathbf{S}$ whose component $S_i(t) \in \{0,1\}$ evaluates to 1 if the neuron i has spiked at any instant in a time window $[t-\tau_w, t]$, where τ_w is the window length, i.e. if $\exists t_f \in [t-\tau_w, t] \mid s_i(t_f) = 1$. Recall $s_i(t)$ refers to the spiking state variable of neuron i. The $\mathbf{S}$ representation becomes natural in Loihi where it is possible to define box PSPs, whereby the synaptic weight w_{ij} is added to u_i when a spike arrives at the i-th neuron and this same value is subtracted after the box duration has passed. We chose this encoding since the $s_i(t) \mapsto S_i(t)$ mapping is essential to the results of [HJM13] and [JHM16] on the exponential convergence of stochastic SNNs to a stationary distribution and its application to CSPs. Formally,

$$S_i(t) = (\rho * s_i)(t) = \sum_{k=0}^{\tau_w} \rho(k) s_i(t-k) \tag{4.1}$$

where the kernel $\rho(t) = H(t-t_f) - H(t-(t_f+\tau_w)) = 1 \forall t \in [t_f, t_f+\tau_w]$ and the $*$ symbol signifies (discrete) convolution.

If a pair of mutually inhibiting neurons are active during the same time interval $[t-\tau_w, t]$, they will conflict, resembling the negative exchange interaction in the spin systems of chapter 2. Thus, one can define the energy of the network by:

$$E = \mathbf{S}^T(t) \cdot \mathbf{W} \cdot \mathbf{S}(t) = \sum_i (S_i \cdot \sum_j W_{ij} \cdot S_j). \tag{4.2}$$

Where T denotes transposition. Notice that because $\mathbf{W}$ and $\mathbf{S}$ are discrete, E is quantised. It would also be possible to use other state variable like v to define E, in which case it would become a real-valued function. This also happens if the weight matrix $\mathcal{W}$ was to be used with time-dependent weights, instead of the adjacency matrix $\mathbf{W}$. However, such high-resolution definitions are of low or no practical use here, where we are interested in spikes and how these mediate the interaction between neurons representing a CSP variable. Furthermore, we use fixed weights for CSPNxNet, so these only have the effect of scaling the energy gaps arising from equation 4.2.

Given the fact that inter-connected winner-takes-all (WTA) circuits compose our networks, we can decompose the total energy of the network in local and global

contributions:

$$E(t) = E_{wta} + E_{CSP} = \mathbf{S}^T(t) \cdot \mathbf{W}_{wta} \cdot \mathbf{S}(t) + \mathbf{S}^T(t) \cdot \mathbf{W}_{CSP} \cdot \mathbf{S}(t) \tag{4.3}$$

Where the matrix $\mathbf{W}_{wta}$ encodes only the WTA connectivity and $\mathbf{W}_{CSP}$ encodes the CSP constraints. This separation is important as we assign, in general, different values for $\mathbf{W}_{wta}$ and $\mathbf{W}_{CSP}$. As we have defined $\mathbf{W}$ and $\mathbf{S}$ to be binary, $E_{CSP}(t)$ returns the number of conflicts between variables existing in the network at time t. Similarly, E_{wta} counts all the conflicts within WTAs. Hence, $E_{CSP}(t)$ captures the solving state while $E_{wta}(t) = 0$ means that all variables have a defined value (or some inactive, see 4.3.6), while $E_{wta}(t) > 0$ is caused by undefined variables. Notice that for dynamic CSPs, in which constraints are created and destroyed, $\mathbf{W}$ also depends on time. Notice that such a change in topology would keep E well defined.

A subtle fact from equation 4.2 is the fact that

$$E = \sum_i \varepsilon_i, \tag{4.4}$$

where $\varepsilon_i = (S_i \cdot \sum_j W_{ij} \cdot S_j)$ counts the number of conflicts that a neuron has with its presynaptic neurons. It is important to highlight that a conflict from the (S_j, S_i) pair only contributes to ε_i on those times were both $S_i = 1$ and $S_j = 1$. We will see later how multicompartment neurons can compute the microscopic components ε_i, allowing an auxiliary neuron to integrate them as E. All in all, we have defined three energy scales for a CSN network, the global or macroscopic energy E, the local or mesoscopic energy E_{wta}, and the neural or microscopic energy ε_i, see figure 4.1. Because multicompartments measure the microscopic contributions and auxiliary neurons can be used to integrate them, our approach brings to light the ability to control, modify and model energy interactions in dynamical systems representable by SNNs.

Interestingly, the fact that $E_{CSP} = E - Ewta$, allows its interpretation as the free energy of the network. In this way, minimising the number of conflicts implies minimising the CSN's free energy. Let us denote by ψ_{sol} a valuation that solves the CSP (assuming a solvable problem), this can thus be found through:

$$\psi_{sol} = \underset{D_i \in \{D_1, \ldots D_N\}}{\arg\min} \; E_{CSP}(\psi_i) := \{D_i \mid D_i \in D \wedge \forall \psi_j \in \Psi : E_{CSP}(\psi_i) \leqslant E_{CSP}(\psi_j)\} \tag{4.5}$$

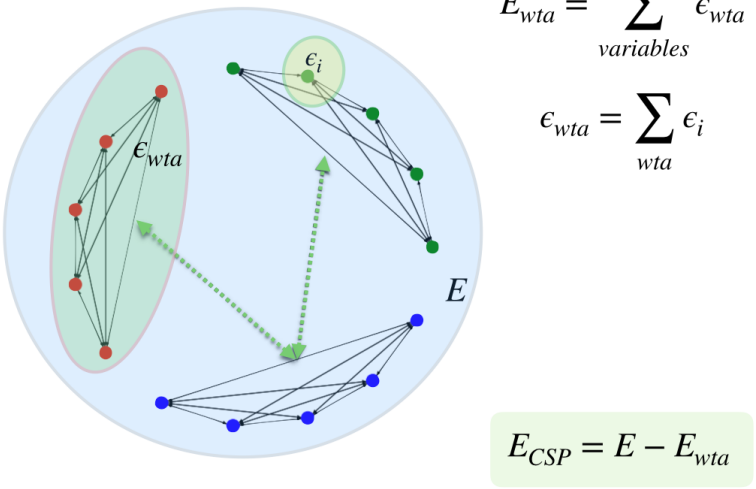


Figure 4.1: Macroscopic, mesoscopic and microscopic energy scales for a CSN network. Three WTA circuits made respectively from red, green and blue neurons are shown. E accounts for the energy of the whole network, E_{wta} is the internal energy from all WTA circuits and ε_i is the energy "stored" in a single neuron.

Where we have used the notation of chapter 2, with Ψ the space of all valuations over the variables defining the CSP, ψ_i denotes a given valuation, and $D = \{D_1, D_2, ..., D_N\}$ is the set of domains over which the $X_i \in \{X_1, X_2, ..., X_N\}$ variables can take values on. The set of constraints $C = \{C_1, C_2, ..., C_M\}$ can be interpreted as prior knowledge [DM04, Jon14]. Hence, equation 4.5 builds up an internal model compatible with such topologically stored knowledge. This interpretation also corresponds to one of the most accepted models of structure and function of circuits of the neocortex as derived from experimental data [DM04]. In problems like Sudoku, the internal model adapts to include the new evidence given by the puzzle cues. With dynamic CSPs, such an interpretation renders the solver extendable to Bayesian frameworks such as *active inference* [FFR+16]. [1].

.

[1] Active inference is generally formulated as free energy minimization (see the free energy principle [Fri10]), such minimization establishes an upper bound on surprise which allows for an estimation of the posterior distribution over hidden states of the world. This chapter solves CSPs as an optimization over a proxy energy function, the extension of our solver then is done by formulating an energy function which is compatible with active inference and the constraints of our neural substrate [FSF+13]

Table 4.1: CSPNxNet Properties for the single compartment and multicompartment segments of the API.

Single Compartment		Multicompartment	
PUBLIC	PRIVATE	PUBLIC	PRIVATE
box length	bias	noise_at_multicompartment	_multicompartments_per_summation
enable noise?	logical core ID	w_ij_exp	_num_summation_neurons
neuron parameters	neurons per core	w_ij_mant	1 to 2 weight
threshold voltage	noise amplitud	w_ij_tot	3 to 2 weight
current decay			bias 2
voltage decay			bias 3
saturation voltage			black_list_num_compartments
negative constraints weight			channel size
positive constraints weight			logical core ID
self excitation weight			threshold voltage 2
WTA excitation weight			threshold voltage 3
WTA inhibition weight			

4.2 CSP Solver with Off-Chip Validation

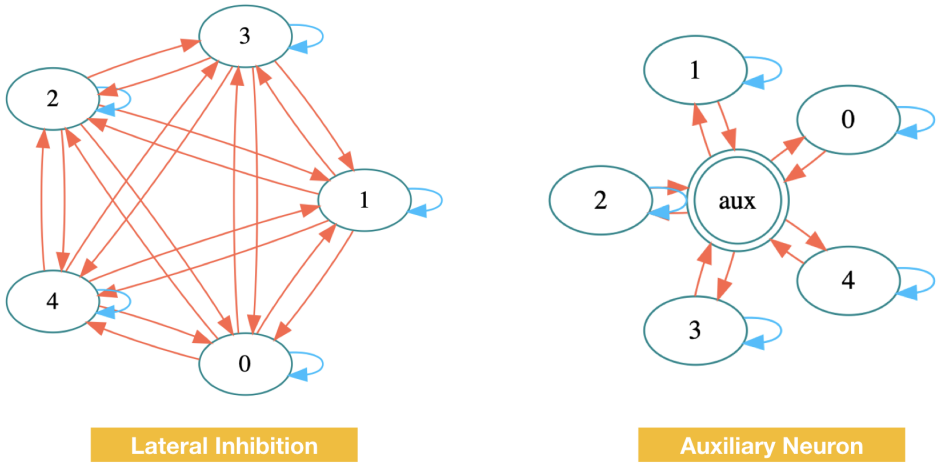


Figure 4.2: WTA connectivity variants.

Let us first implement a CSP solver whose functionality is equivalent to that of sPyNNCSP. This means we probe the spiking state of every neuron to get $\mathbf{s}(n \cdot \Delta t)$ with $\Delta t \geqslant 1$ and compute $E(n \cdot \Delta t) = \mathbf{S}^T(n \cdot \Delta t) \cdot \mathbf{W} \cdot \mathbf{S}(n \cdot \Delta t)$ on host (post-processing) to recover the progress of the stochastic search, finding out if the CSP was solved ($E = 0$).

4.2.1 API Design

We entirely based the off-chip-validation solver on Loihi's highest level API, NxNet. Similar to sPyNNCSP, the frontend of CspNxNet allows the user to define the problem through the tuple $\langle X, D, C \rangle$ and control the stochastic search through the network and neuron parameters. From this minimal information, CspNxNet will encode the problem into a stochastic SNN representation where the dynamics of the network will implement a stochastic search over the combinatorial space of the problem.

The solver itself corresponds to an instantiation of the CspNet class, which is the core of the API we call CSPNxNet. Its design is shown in figure 4.3 with the class properties listed in table 4.1. To ease visualisation, we have grouped the methods in figure 4.3 according to their role, set as the headers in a purple background. Private and public methods have been colour-coded yellow and blue, respectively. In this way, the cells with blue background correspond to the frontend which the app developers or direct users will mostly employ, while the yellow cells present the internal methods for the API development. These latter build the CSN, any internal process for its functionality and generate the automated parameterisations. From the front-end, the shortest use to solve a CSP will be:

```
# Create solver instance
net = CspNet( number of variables , domains size , constraints , runtime)
# Setup probes for network activity
spikes , voltage , current = net.setup_network_probes(probe_spikes=True ,
                                                      probe_voltage=True ,
                                                      probe_current=True)

# Run simulation
net.run()

#Get solution
net.offline_get_solution
```

Table 4.2: Constructor attributes for the single compartment CSPNxNet API.

CSP	TOPOLOGY	NEURON	SYNAPSE	SIMULATION
number_of_variables	self_excitation	params_of_principal_neurons	w_wta_exc	runtime
states_per_variable	directed_graph	box_duration	w_wta_inh	do_setup
neurons_per_state	wta_type	bias_to_fire	w_constraints_inh	
inh_constraints		spike_to_fire	w_self_exc	
exc_constraints		randomized_seeds	w_exp	
clamped_values		randomize_v_init		
		enable_noise		
		randomize_v_interval		
		seed		
		vMinExp		
		v_th_1_mant		

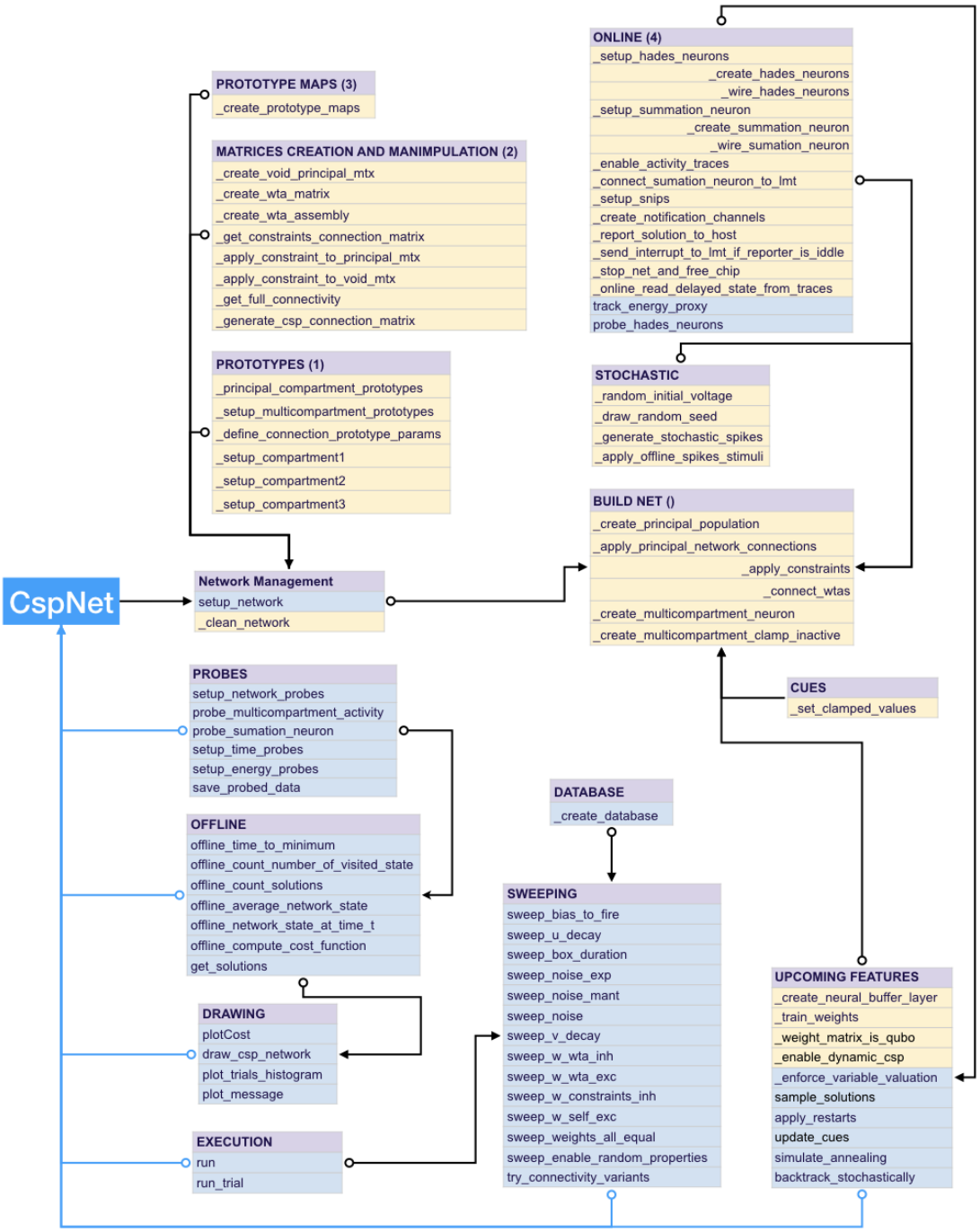


Figure 4.3: CSPNxNet API design.

```
# Plot activity
figure()
spikes.plot()
voltage.plot()
```

```
current.plot()
```

If needed, the user can define further attributes from the constructor. We list the available options in table 4.2. These define the CSP instance, the network topology, neuron parameters, synaptic connections, and the simulation itself. Following figure 4.3, when the user creates a CspNet object, the API executes the *setup_network* function, and this will then walk the backend creating the necessary dependencies according to the constructor attributes as follows:

Create prototypes. Neurons and synapses in NxNet are defined through compartment prototypes. This way of programming helps the compiler to allocate the hardware resources from Loihi efficiently, seizing any redundancy in the network specification. The compartment prototypes will encode all properties of the CUBA LiF neuron (equations 3.10 and 3.9) and thus define the intrinsic dynamics of the network, e.g. average firing rate and noise level. Connection prototypes encode the synaptic interaction between neurons; these include weights, delays and specification of the postsynaptic potentials types and parameters.

Compute adjacency matrix. In this API we encode all connectivity of the principal network in a single 2-dimensional Numpy array W of size $\mathcal{N} \times \mathcal{N}$, where $\mathcal{N}$ is the total number of neurons in the principal SNN. W encodes both the CSP adjacency matrix as well as the WTA connectivity.

Define prototype maps. The neurons in the network correspond to an N-array of NxNet objects, the neural prototypes should be assigned to each object in the array depending on whether the neuron is a cue, a cue complement, a principal neuron or an auxiliary neuron of a WTA circuit.

Build the network. the setup network function then uses the network specification to create the single compartment neurons and the synaptic connections between them so that the SNN corresponds to a CSN.

As part of the network specification, we want to drive the network dynamics as a stochastic process in which every neuron fires randomly with some average firing rate unless inhibited by the winner neurons. In sPyNNCSP, we used on-chip in-software spike sources obeying a Poisson process (2), these acted as auxiliary neurons, which where presynaptic to every principal neuron. Although spike injection is available in Loihi, we do not use it to drive the network dynamics. It is equivalent to the *SpikeSourceArray* method in PyNN which requires the spike times to be defined beforehand

on the host and transferred to the chip, creating unnecessary overhead. Instead, each neuron is driven by a bias current b_i and we include stochasticity by using Loihi's randomisation of state variables (section 3.2.4). The chip allows the efficient randomisation of either the compartment voltage, compartment current or refractory period by adding a pseudo-random number η (equation 3.11) to any of these. This slight modification of not using Poisson spike sources halves the number of neurons required to drive the network dynamics in CspNxNet compared to SPyNNCSP. The noise level is modified through the definition of *noiseExp* and *noiseMant* in equation 3.11.

If the voltage is randomised, the neural dynamics for the off-chip validation version of the solver is given by:

$$u_i(t) = \sum_{j \neq i} w_{j,i} \left(\frac{e^{-t/\tau_u}}{\tau_u} \Theta(t) * \mathbf{s}_j \right)(t) + b_i, \tag{4.6}$$

$$\dot{v}_i(t) = -\frac{1}{\tau_v} v_i(t) + u_i(t) - \theta_i \mathbf{s}_i(t) + \eta_i(t), \tag{4.7}$$

$$s_i(t) = v_i(t) > \theta_i \tag{4.8}$$

$$v_i(t)[s_i(t)] = 0 \tag{4.9}$$

Notice we used the exponential PSP as it is the one used for sPyNNCSP and there is no need here for box PSP. We will see later this is not the case for the on-chip validation design.

Besides the noise amplitude

$$|\eta| = 255 * 2^{(-7+noiseExpAtCompartment)}, \tag{4.10}$$

other straightforward ways to control the neural dynamics are through the θ_i/b_i quotient, which regulates the average firing rate of each neuron, and the θ_i/w_{ij} quotient, which controls the amplitude of the inhibition with respect to the threshold value. In turn, the intra-WTA and inter-WTA weights, w_{wta} and w_{cons}, define the strength of the interaction between neurons in the same and different WTAs respectively. In developing CSPNxNet, we have kept constant weights making difference only according to their modality, i.e. excitatory and inhibitory inter-WTA, excitatory and inhibitory intra-WTA, and self excitation which are in general different, we denote such weights by w_{wta_inh}, w_{wta_exc}, w_{cons_inh}, w_{cons_exc} and w_{self} respectively.

To represent variables, we offer support for two different WTA motifs shown

in figure 4.2. The first, ("lateral"), is based on an all-to-all lateral inhibition, as in sPyNNCSP, and the second ("aux") uses an auxiliary neuron to mediate competition. In the latter, every principal neuron in the WTA excites the auxiliary neuron, which in turn inhibits every principal neuron. Thus the first neuron to fire will exert indirect inhibition on the other. The *aux* type can be more economic in space resources, this is because its number of synapses grows linearly with the domain size as $2D$, or $3D$ if self-excitation is enabled, whilst the *lateral* motif grows quadratic as $(D chose 2) = (D^2 - D)/2$, or $(D^2 + D)/2$ if self-excitation is enabled (see table 4.3) [2]. The *lateral* motif, however, saves one neuron per variable and is trivial to program, so it can be useful in scenarios where neurons are scarce, constraints density is low, and domain size is small (see figure 4.4). Consider a variable assuming 50 possible values. It would require 1225 synapses and 50 neurons for the *lateral* motif and 100 synapses and 51 neurons for the *aux* motif. So in this case *aux* is preferable. However, if we are dealing with 50 binary variables, these require (disregarding constraints) 100 neurons and 100 synapses for the *lateral* motif, and 150 neurons with 200 synapses for the *aux* motif. So, in this case, the *lateral* version is a better choice.

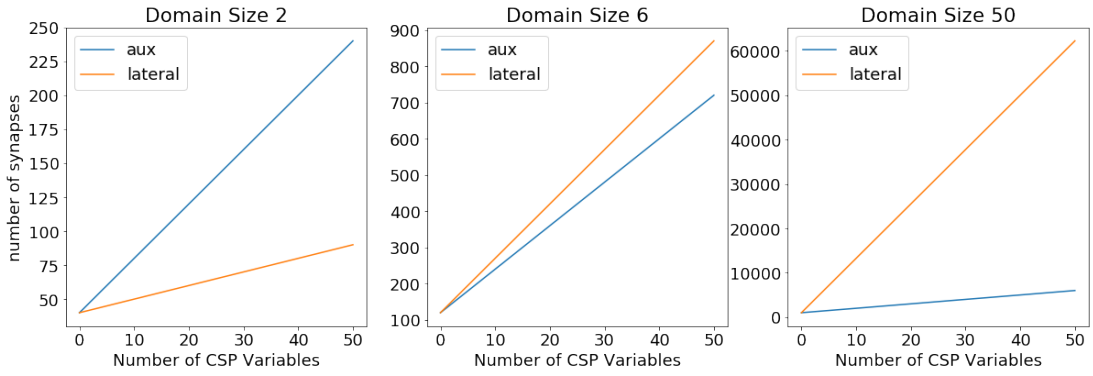


Figure 4.4: Dependence of the number of synapses on number of variables for WTA motifs for various domain sizes. For bianary variables (left), the "lateral" motif is better, but for large domain sizes "aux" is largely superior (right).

The API design could easily accept several neurons per domain per variable as it was the case for sPyNNCSP. However, in the spirit of using minimal resources, targeting best performance and efficiency, we kept a single neuron to represent each domain in

[2]This analysis regards the spatial resources of interest in neuromorphic hardware, i.e., neuron count and synapse density. Regarding time complexity, the fine-grained parallelism in Loihi attenuates the difference between *aux* and *lateral* motifs, nevertheless, as neurons are updated sequentially inside a neurocore, the increase in neurons count implies a slight increase in time, but this increase is sublinear on the network size, in contrast to the respective exponential growth of the state space that has to be searched.

Motif	Aux				Lateral			
Network	WTA		CSN		WTA		CSN	
self-excitation	no	yes	no	yes	no	yes	no	yes
Synapses	$2D$	$3D$	$2DV+2DC$	$3DV+2DC$	$\frac{D^2-D}{2}$	$\frac{D^2+D}{2}$	$(\frac{D^2-D}{2})V+2DC$	$(\frac{D^2+D}{2})V+2DC$
Neurons	$D+1$		$(D+1)V$		D		DV	

Table 4.3: Space complexity for WTA and CSN networks as a function of domain size (D), number of variables (V) and number of constraints (C). It is assumed all variables have the same domain size and the CSP is linear.

every WTA unless stability was affected, which has not been the case, recall in Loihi we randomise the state variables instead of applying stochastic spikes as input. So, all our experiments were performed with one neuron per domain in the WTA encoding a CSP variable. Recall SpyNNCSP required ≈ 25 neurons per domain, so this choice represents a further 25-fold saving in neurons count.

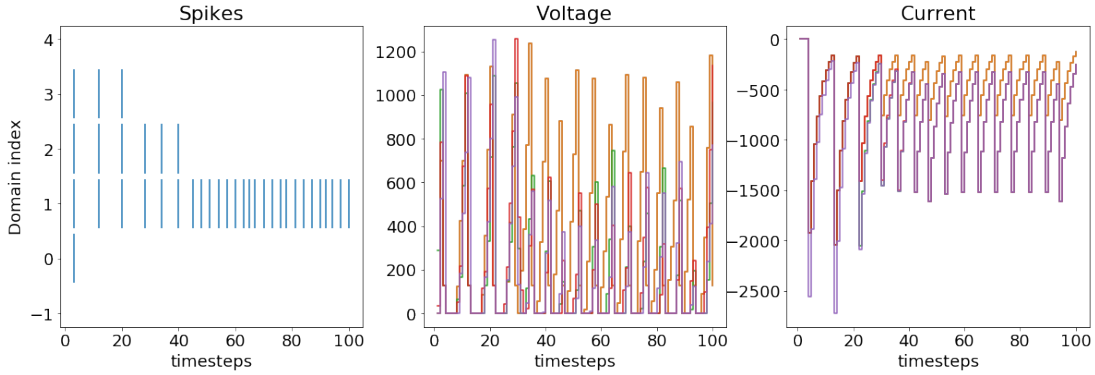


Figure 4.5: 5-neuron WTA run with the NxNetCSP API. The yellow current and voltage traces correspond to the winner neuron with domain index 1.

We show in figure 4.5 the spiking activity, compartment voltage and compartment current for a 5-neuron WTA running in Loihi, i.e. *CspNet(variables=1, domine size =5)*. We have set the parameters so that some competition is evident, notice that four neurons are active until a winner is chosen after the 40^{th} timestep.

To implement the CSN, we use synaptic connections between WTAs. In the case of a non-equal constraint $X_l \neq X_m$ (figure 2.1) for example, the respective neurons encoding the same domains in the WTA_l and WTA_m compete through mutual inhibition, in the same fashion as chapter 2. To illustrate such construction, let us consider a simple spin chain defined by 10 sequential binary variables with first-neighbours interacting with non-equal constraints. This can be created and solved by executing:

```
net = CspNet(variables=10, domine size =2, constraints=[(i,i+1) for i in range(0,9)])
spikes, voltage, current = net.setup_network_probes()
```

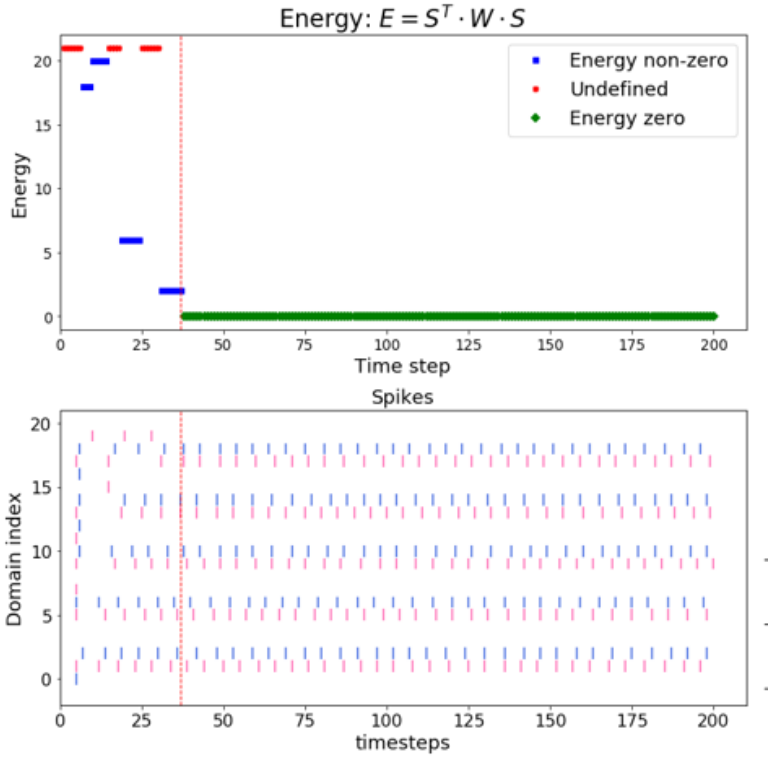


Figure 4.6: Chain of 10 spins with anti-ferromagnetic interaction. The red vertical line indicates the $E = 0$, which occurs at the timestep 37. In the spikes plot, red and blue represent up and down states respectively. The transition between exploration and stable dynamics is evident at t=37.

```
net.run()

cost=net.offline_compute_cost_function()

figure()
spikes.plot()
current.plot()
net.plotCost(cost)
```

From which we obtain figure 4.6. In the spikes diagram, it is easy to observe an initial period of competition lasting the first 35 timesteps, after which the network converges to a state satisfying all constraints, the winner neurons remain active, keeping the losers inactive. The Energy plot corresponds to the off-chip computation of equation 4.2, where the green dots reflect zero $E = 0$, which corresponds to the solution of the CSP, blue dots are those states that are well defined but violate some constraints while red dots correspond to states where some variables are undefined, either because are multi-valued or because remain inactive.

The dynamics of the network is controllable by a set of free parameters, namely

the arguments to the compartment prototypes and the weights for the synapses. To alleviate the process of parameter tuning, we implemented the set of methods labelled with the *sweeping* role in figure 4.3. Each of these calls the *_create_database* private method which runs several trials of the network, setting for each time a different value of the swept parameter, in a range defined by the user. The results of each run: probed data, postprocessing and metadata, are optionally recorded into an HDF5 file. This method also creates a text file, if it does not exist yet, where the timestamp for calls to the corresponding sweeping method will be collected followed by a summary of parameter combinations that generated solutions, as well as their time to solution. It is allowed to nest the sweeping methods so that a single call can potentially sweep all parameters. For now, this results in a sequential assignment of the parameters, a kind of brute force protocol. In the following version of the API, the sweeping will adapt according to the results of the last run. It will optimise for two essential features of the CSN, namely, the WTA behaviour for each variable and a trade-off between exploration and exploitation. Notice that such parameter fitting constitutes in itself an optimisation problem, however in a much lower dimension than the original CSP. It is worth noting that the number of free parameters does not depend on the network size, resulting in a time complexity $O(1)$ for parameter tuning. Ultimately, these methods are just the building blocks for sophisticated CSP solving strategies to be run alongside the online solver and from the embedded LMT processors. Using SNIPs, these will coordinate the search so that fluctuations in ε_i are enough to scape local minima while keeping a bounded network dynamics (moderate spike traffic).

4.2.2 Solution to CSPs with CSPNxNet and Off-chip Validation

We now use the API to solve the world map colouring (WMC) problem with four colours, as well as the Sudoku puzzle. We show the results for both problems in figures 4.7 and 4.8 respectively. We can see that in both cases, the network finds a solution and stabilises in the solving configurations for Sudoku and degenerate solutions for WMC. When cues are included in the problem formulation, as is the case of the Sudoku puzzles, the connectivity is created to keep arc consistency similar to a filtering algorithm [Rég04], so the API keeps only synapses consistent with the constraints. It is possible to clamp variables with strong bias current on the desired domain while having no bias on the others. However, in problems were excitatory constraints exist such method could lead to change in the variable value and hence, arc consistency is preferred.

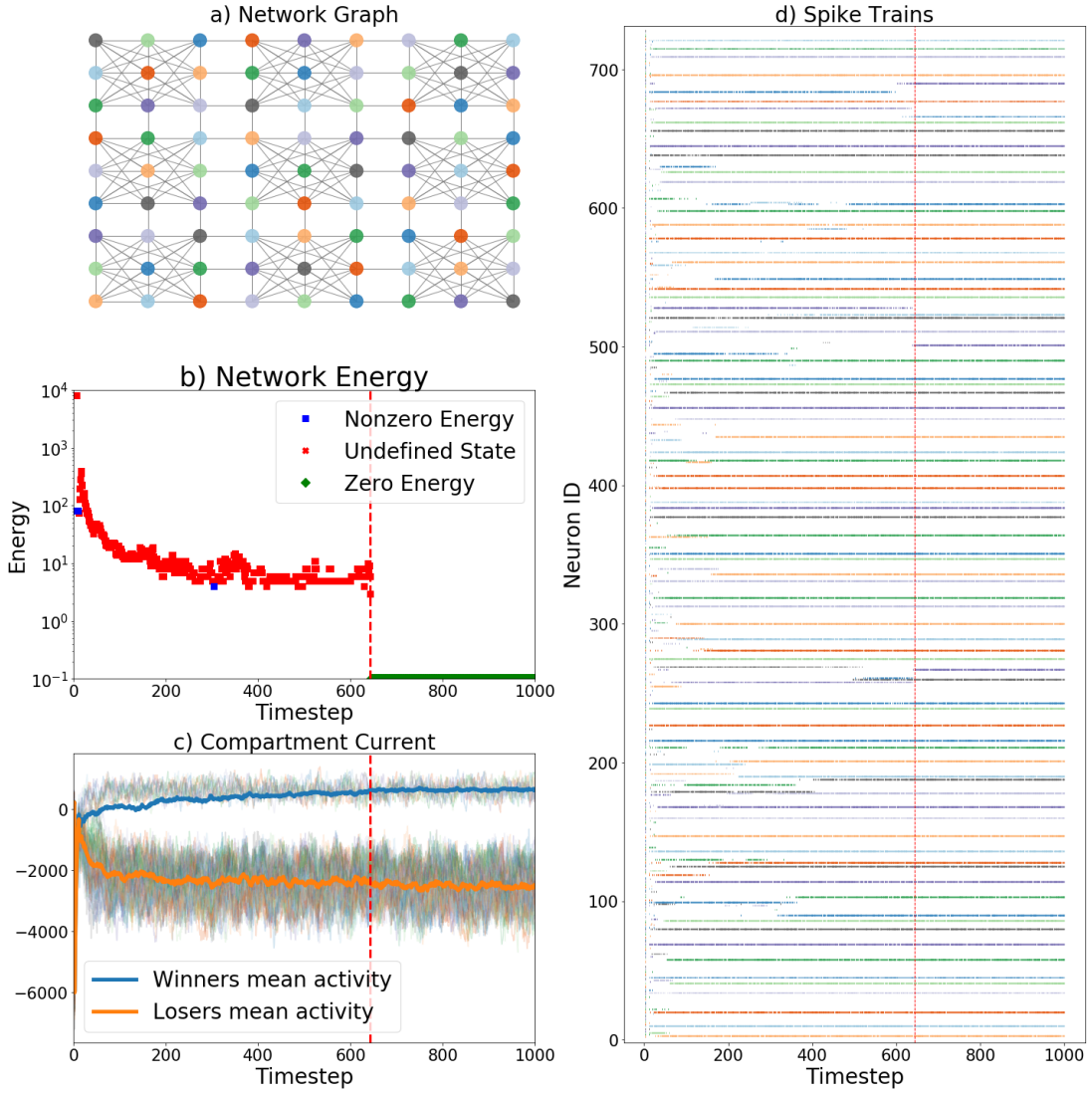


Figure 4.7: The solution to a 9×9 sudoku puzzle using the CSPNxNet API. a) Solved CSP network. Nodes, colours and edges represent WTAs, winner domains and constraints respectively. b) evolution of the cost function, $E = 0$ when all constraints are satisfied. Undefined states are those with some inactive or multivalued variables. c) Evolution of compartment currents from all compartments in the network. The mean activity for winning and losing compartments is also shown. Competition continues until a solution is found at timestep 643. d) Spiking activity of the 729 neurons. The dashed red vertical line across plots indicates the timestep at which a solution was found.

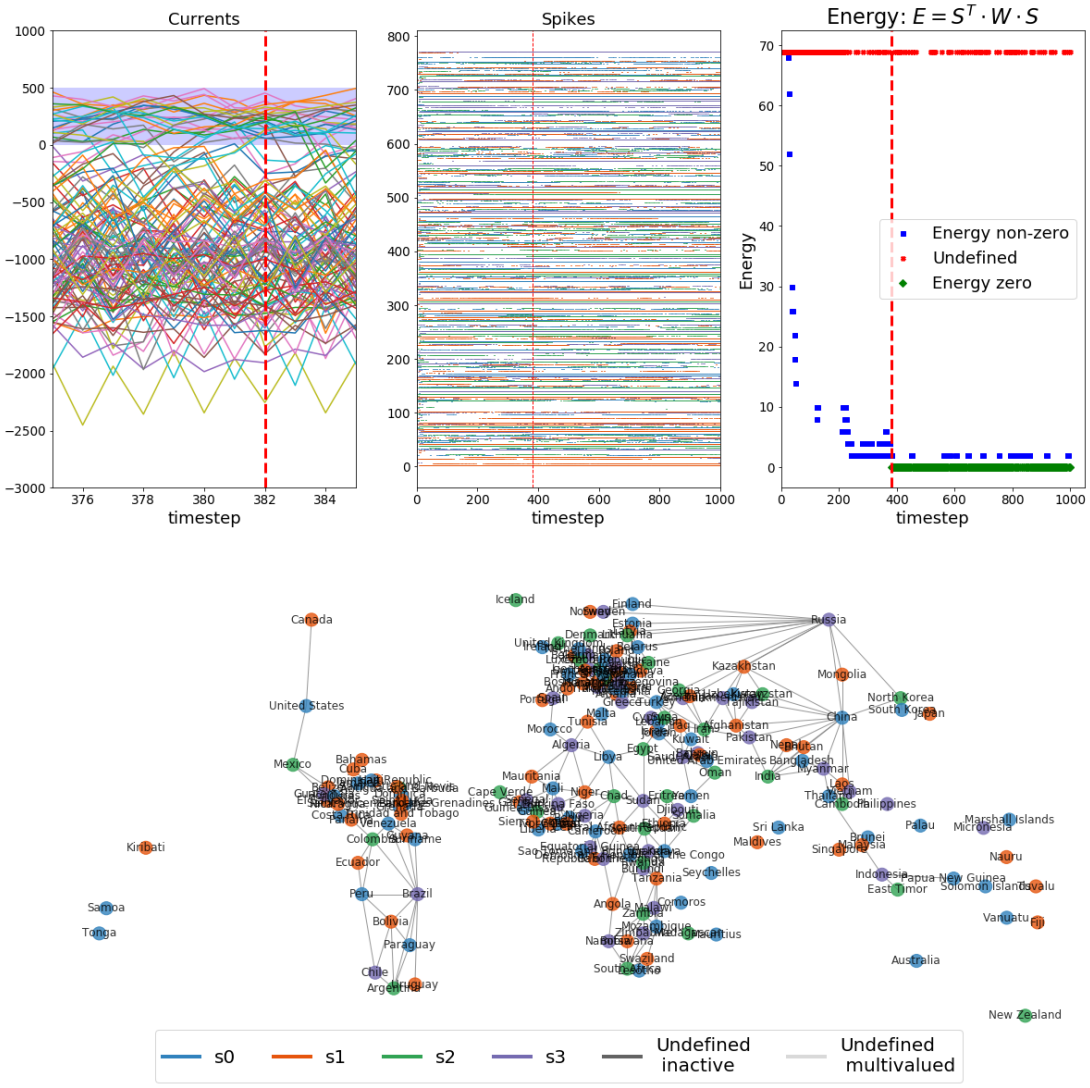


Figure 4.8: Solution to the 4-coloring of the map of the world (193 states recognised by the United Nations). a) Current traces for network compartments showing the mean activity for winning and losing neurons. b) Spike trains of the network compartments, every four rows correspond to the domains of a variable, colors code for domains in a variable. c) Network energy showing the evolution of the stochastic search until satisfaction is achieved. d) Network graph, nodes represent WTAs, colors code the valuations and edges corresponds to constraints between variables.

Recall that Loihi has a variable duration for its real timestep, and this is the reason for presenting timesteps on the x-axis of figures 4.7. Probing the network activity results in considerable overhead. On average, the time per timestep for the problems above will take between 10-30 μs when no probes are used. However when probes are activated, the average time per timestep grows by one order of magnitude, i.e., $approx300\mu s$ The 4-colouring of the world map solved at timestep 382, which means a solving time of only $\approx 114,6ms$. This same problem required 123 s when using the sPyNNCSP API. For Sudoku, the solving time was 643 timesteps resulting in $\approx 192.9ms$, this problem took an average of 1.31 s in sPyNNCSP.

Regarding the number of neurons, we have pointed out how we save 50 % by not using poison spike sources. Furthermore, the networks are stable and even converge to a stable solution while using only one neuron per domain, instead of the 25 for Sudoku and 50 for the WMC in chapter 2. Thus, in CSPNxNet, we require 50 to 100-fold fewer neurons, which amounts for two orders of magnitude improvement for the WMC problem. Fewer neurons imply a dramatic reduction in the number of synapses, which are 3614 and 17496 for WMC and Sudoku, respectively, to be compared with table 2.1. Thus, the model becomes portable and usable in edge devices.

Probing a high dimensional $\mathbf{S}(t)$ is slow and resource-intensive. Furthermore, the off-chip validator may miss a solution when $\Delta t > 1$. Hence, In the following section, we demonstrate our method for performing the on-chip measurement of E, as well as the detection of $E = 0$.

4.3 CSP Solver with On-chip Validation

One of the important challenges when developing applications for neuromorphics is the need to satisfy the locality constraint by which the network state is not accessible to its components [LWC+18a]. Locality here is a consequence of the distributed memory and processing which remain close to each other across the machine, greatly reducing both power consumption and execution times [Fur16a]. A further challenge for SNNs in Loihi arises from the bottleneck caused by recording each state variable of every neuron and their subsequent transfer to the host server for postprocessing. Loihi is instead intended to make SNN-based computations and take decisions with spatially and temporally sparse interaction with the external devices and actuators. Its communication with the external world would be ideally limited to sense and integrate input data and then retrieve the output of its computation or inference. These facts imply that the

network or neuron architectures should be self-validating and autonomously make decisions of when to communicate with the external ecosystem, i.e., robot components, humans, nature or other computing devices.

In what follows, we present our strategy to endow spiking neurons with the ability to report their state of conflict with the rest of the network to an integration neuron. This latter acts as a trigger for decision making as specified either by arbitrary neural interfacing processes executed on-chip by the x86 cores or by other SNNs existing across the neurocores.

4.3.1 Self-Verifying Network

To eliminate the need for expensive probing of each neuron, as well as postprocessing, the network should evaluate at runtime and in-parallel its energy function E as defined by equation 4.2. Whenever $E = 0$, the network should notify that a satisfying state was found at time t_z so that we can read the solution by extracting either $\mathbf{s}(t_z)$, $\mathbf{v}(t_z)$ or $\mathbf{u}(t_z)$. In this scenario, the goal is that no information leaves the chip or is recorded by it unless a solution exists encoded in the instantaneous network state. Thus, eliminating the overhead incurred by storing and transferring data between the components of the heterogeneous computing stack, or by recording unnecessarily large amounts of data. We achieve such functionality by leveraging the the multicompartment features of the chip (section 4.3.2). For the following, let us recall that W denotes the binary adjacency matrix while $\mathcal{W}$ corresponds to the weight matrix, also, $s_i(t)$ denotes the spiking state variable of neuron i, whilst S_i is the i-th component of the network vector state resulting from extending s_i through τ_w timesteps, which is a way of considering the lasting effect of the spike on the network.

Let us begin with the definition of E from equation 4.2, were we interpreted the interactions in an SNN in analogy with the energy stored in a system of interacting particles:

$$E = \sum_i (S_i \cdot \sum_j W_{ij} \cdot S_j) \tag{4.11}$$

Realise that the term $\sum_j W_{ij} \cdot S_j \in [-N_{pre}, 0]$ on the right hand side of equation 4.11, where N_{pre} is the number of presynaptic neurons to i, can be retrieved from the compartment current $u(t+1)$ in equation 3.3, which for a box PSP of lenght τ_w can be

written as

$$u(t) = \sum_j \sum_{k=0}^{\tau_w} w_{ij} \cdot \rho(k) \cdot s_j(t-k) \tag{4.12}$$

$$= \sum_j w_{ij}(\rho * s_i)(t)$$

$$= \sum_j w_{ij} S_j(t). \tag{4.13}$$

Moreover, we can obtain the adjacency matrix W by binarizing the weight matrix $\mathcal{W}$ over non-zero weights:

$$\sum_j W_{ij} \cdot S_j = \sum_j [w_{ij} \neq 0] S_j$$

$$= \sum_j [w_{ij} S_j \neq 0], \tag{4.14}$$

where the expression inside square brackets outputs a truth value. In the particular case of equal and constant weights of magnitude $|w|$ across the network, one can write

$$\sum_j [w_{ij} S_j \neq 0] = \frac{1}{|w|} \sum_j w_{ij} S_j. \tag{4.15}$$

So equation 4.14 can be expressed as

$$\sum_j W_{ij} S_j = \frac{u_i(t)}{|w|} := \bar{u}_i(t) \in [-N_{pre}, 0]. \tag{4.16}$$

which evaluates different of zero when any neuron j which is presynaptic to i fires. We can then rewrite 4.11 as

$$E(t) = \mathbf{S}^T(t) \cdot \bar{\mathbf{u}}(t+1) \tag{4.17}$$

$$= \sum_i S_i(t) \cdot \bar{u}_i(t+1). \tag{4.18}$$

Following equation 4.4,

$$\varepsilon_i(t-1) = S_i(t-1) \cdot \bar{u}_i(t). \tag{4.19}$$

$$\tag{4.20}$$

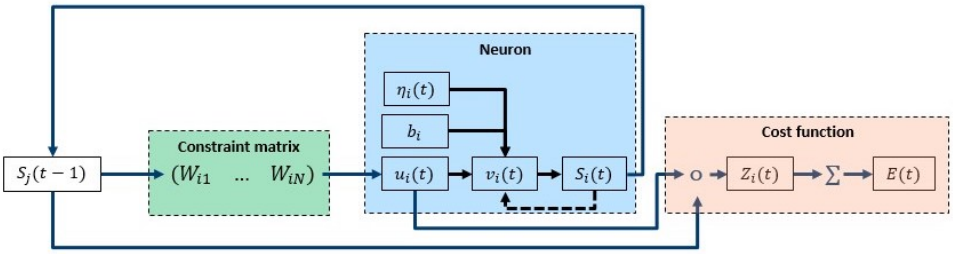


Figure 4.9: Ideal architecture to compute E from local information. Spikes $s_j(t-1)$ from all presynaptic neurons are integrated by neuron i (blue box) through the constraints matrix $\mathcal{W}$ (green) into u_i. In turn, v_i absorbs u_i, is randomised by η_i and driven by b_i, eventually causing a spike. When this happens, the local energy contributions $S_j(t-1) \cdot u_i(t)$ are transferred to a Σ neuron which integrates them into E.

This means that the intrinsic dynamics of each neuron already computes the components of their local contribution ε_i to E, which can be regarded as the neuron's intrinsic energy (figure 4.1) when the network is in a particular configuration $\psi_n = \psi(n\Delta t)$. If each neuron can internally compute the local cost $\varepsilon_i(t)$, we can set up the summation by a neuron Σ (summation neuron) which is postsynaptic to all principal neurons. To this point, the basic architecture is shown in figure 4.9,

We still need to compute ε_i, however, the $S_i(t-1) \cdot u_i(t)$ element-wise multiplication is not directly supported by Loihi and we need to make $S_i(t-1)$ available to the neuron i at time t.

4.3.2 Multicompartment Computation and Integration of ε_i

To obtain $\varepsilon_i(t)$ one needs to have $u_i(t)$ and $S_i(t-1)$ internally available to the neuron which would need to output their dot product. We achieve this by using Loihi's ability to group compartments as multicompartment neurons, allowing the propagation of u_i from the dendrite to the soma to the axon.

In order to build multicompartments, each neurocore has a memory stack of depth 2. Any state variable from a compartment can then be pushed into the stack and be used by the next compartment while the runtime software is advancing the computations for one timestep. Every compartment can execute a small set of operations between its internal state and the information saved on the stack, (see multicompartment operations in section 3.2.3). Given the fact that the u_i and S_i of interest belong to different time references, we make $S_i(t-1)$ available to the neuron by creating an autapse (a

synapse from the neuron onto itself [VDLG72]) with 100% transmission probability. This makes $S_i(t-1)$ available at time t in the dendritic node where $u_i(t)$ resides, see figure 4.10. Autapses are very common in biological neural networks, both in-vivo and in-vitro [TBS97, PM99, BS91]. There is, however, no full understanding of their role for the brain's information processing. These are shown to mediate anticipated synchronisation [PRM19], promote specific neural oscillations [HK04] or even the kind of synchronisation underlying conditions such as epilepsy [FWW$^+$18]. Here, we show a further possible role of autapses for SNN architectures, which beyond any biological plausibility, is useful as a computational resource in SNNs design. Note that besides enabling operations on state variables, the multicompartment structure is necessary to avoid interference between the underlying CSP solver dynamics and the computation of E.

The multicompartment model (figure 4.10 top) begins with using a soma compartment ($\mathcal{B}$ for *Base* compartment, purple block), which performs the normal role of the neuron in the SNN from the off-chip CSP solver. It receives external synapses corresponding to the adjacency matrix (green box), and interacts with other $\mathcal{B}$s in the network through its axon arbour. In parallel, the soma also pushes $u(t)$ into the stack. On top of the soma, the neuron has two compartments, which, because $\mathcal{B}$ emits spikes and receives the external synaptic input, might be more accurately referred to as axonic compartments, $\mathcal{M}$ (for *Middle* compartment, yellow box) and $\mathcal{T}$ (for *Top* compartment dark green). The neuron forms an axo-axonic autapse from $\mathcal{B}$ onto $\mathcal{M}$ to propagate $S_i(t-1)$ as explained above. Although we do not drive our implementation by biological realism, it is important to highlight that such autaptic connectivity is not at all biologically implausible. Autapses exist in a diversity of forms, axodendritic, dendo-dendritic and axo-axonic as observed in both cortex, hippocampus and cerebellum [TBS97, BS91, Bek02, PM98, PM99]. The $\mathcal{M}$-$\mathcal{B}$ connection functionally replicates the firing state of the soma in $\mathcal{M}$, which pushes its spiking variable into the stack. $\mathcal{T}$ goes onto pulling both $u_i(t)$ and $S_i(t-1)$ from the stack and computes $\hat{\varepsilon}_i(t) = S_i(t) \wedge u_i(t+1)$. Where the *AND* operator ($\wedge$) emulates a dot product. Note that $\hat{\varepsilon}_i \in [-1,0]$ while $\varepsilon_i \in [-N_{pre},0]$. Hence, we use $\hat{\varepsilon}_i$ as a proxy for ε_i which satisfies:

$$\begin{aligned} \hat{\varepsilon}_i = 0 &\iff \varepsilon_i = 0 \\ &\& \\ \hat{\varepsilon}_i \neq 0 &\iff \varepsilon_i \neq 0 \end{aligned} \tag{4.21}$$

While we have lost information with $\hat{\varepsilon}_i$ about how many neurons conflict with neuron i, we have preserved information about if conflicts exist at all or not. Hence, the coarse resolution in relation 4.21 is sufficient condition for the on-chip validation framework to remain valid.

Initially, we implemented the 4-compartment neuron model shown in (figure 4.10 bottom) because only v_i could be pushed into the multicompartment stack. In that case, we needed an additional dendrite compartment $(\mathcal{D})$ on the bottom (blue box), which receives the synaptic input from other neurons, integrates them into u_i and passes this value through the stack to $\mathcal{B}$. This latter integrates u_i into v_i, also incorporating the bias current and noise. Some extra difficulties come with this approach. As described in the previous chapter, the same LFSR is used to randomise the voltage of all compartments in a neurocore, this, in turn, implies a single specification of noise amplitude per neurocore. Hence, when pushing v_i into the stack to be absorbed by $\mathcal{T}$, the neuron is not only propagating u_i but also η_i. Throughout development, the ability to push u_i became available and with it the fusion of the functionality of the two first compartments. Except for the unintended propagation of η_i, the behaviour of $\mathcal{D}+\mathcal{B}$, in the 4-compartment model, is equivalent to the behaviour described above for the $\mathcal{B}$ of the 3-compartment model with the additional benefit of saving one compartment per neuron.

4.3.3 3-Multicompartment Model

In this section we formalise the model used to create the neuron $(\mathcal{B},\mathcal{M},\mathcal{T})$. An important difference with the off-chip validation solver is the requirement to use the box synaptic kernel ρ. By design, the box PSP requires $\delta_u = 0$, with the compartment current given by

$$u_i(t) = \sum_j w_{ji} \sum_{k=0}^{\tau_w} \rho(k) s_j(t-k) \tag{4.22}$$

This also implies that the on-chip validation solver needs to use the randomization of the compartment voltage. Randomising either u_i or the refractory delay would interfere with the definition of the box PSP. Then,

$$v_i(t) = v_i(t-1)(1-\frac{\delta_{v,i}}{\delta_{max}}) + u_i(t) + b_i + \eta_i. \tag{4.23}$$

Here, $\delta_{max} \mapsto \infty$ is the maximum decay allowed by the chip.

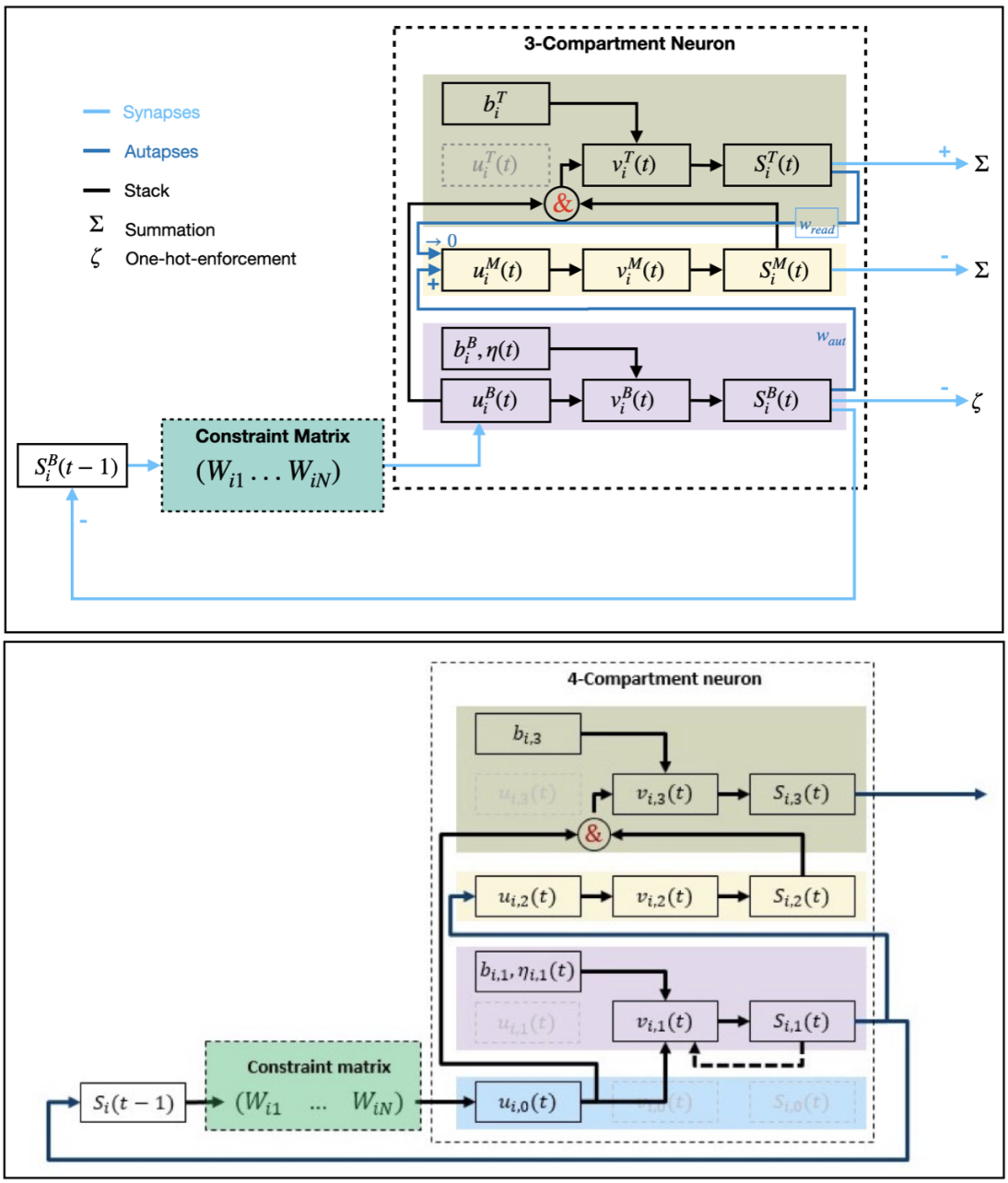


Figure 4.10: Multicomaprtment architecture, flow of state variables and their transformation across the 3-compartment (top) and 4-compartment (bottom) neuron models. Both models are functionally equivalent so the smallest is preferred.

Soma Compartment. Every $\mathcal{B}_i$ correspond to an integrate and fire neuron (IF) driven by its bias current $b_i^{u,\mathcal{B}}$, and receiving inhibitory input from other $\mathcal{B}_j$ with a weight w_{ij} and a box PSP. $\mathcal{B}_i$ fires whenever its voltage surpasses the threshold $\theta_i^{\mathcal{B}}$. This

compartment can be summarised trhough:

$$
\begin{aligned}
b_i^{\mathcal{B}} &\neq 0 \mapsto f.p. \\
\theta_i^{\mathcal{B}} &\neq 0 \mapsto f.p. \\
w_{ji} &\leqslant 0 \mapsto f.p. \\
\delta_u^{\mathcal{B}} &= \delta_v^{\mathcal{B}} = 0 \\
S_i^{\mathcal{B}}(t) &= [v_i^{\mathcal{B}} > \theta_i^{\mathcal{B}}] \\
u_i^{\mathcal{B}}(t) &= \sum_j w_{ji} \left[\sum_{k=0}^{\tau_w} \rho(k) s_j(t-k) \right] \\
v_i^{\mathcal{B}}(t) &= u_i^{\mathcal{B}}(t) + \eta_i^{\mathcal{B}}(t).
\end{aligned}
\tag{4.24}
$$

Where $f.p.$ labels the free parameters, which in this case are $b_i^{\mathcal{B}}$, $\theta_i^{\mathcal{B}}$ and w_{ji}. Regarding multicompartment operations, $\mathcal{B}$ pushes $u_i^{\mathcal{B}}$ into the stack which will be absorbed by $\mathcal{T}$. It also sends $S_i^{\mathcal{B}}(t)$ to $\mathcal{M}$ through an excitatory autapse with weight w_{aut}.

Axon Compartment $\mathcal{M}$. $\mathcal{M}$ receives $S_i^{\mathcal{B}}(t)$ at time $t+1$ and integrates it into $u_i^{\mathcal{M}}(t)$ with a box PSP. This compartment acts as a relay station which pushes $S_i^{\mathcal{B}}(t-1)$ into the stack for later use by $\mathcal{T}$. Thus, $\mathcal{M}$ should respond to every spike from $\mathcal{B}$ with a spike burst which lasts τ_w, rather than a single spike or box pulse, otherwise the neuron will not detect violations happening in a timestep resolution. We satisfy such a condition by setting $\delta_i^{\mathcal{M}}$ to its maximum value $\delta_{max} \mapsto \infty$. In this way, $u_i^{\mathcal{M}}(t)$ will stay up during τ_w, but $v_i^{\mathcal{M}}(t)$ will decay back to zero after every spike at which point it integrates $u_i^{\mathcal{M}}(t)$ again, resulting in an interspike interval of one timestep during the interval $[t_f, t_f + \tau_w]$. Our multi-compartment neuron model also includes an autapse from $\mathcal{T}$ to $\mathcal{M}$ characterised by a very small weight w_{read}. We use this autapse for delayed readout of the network state as explained later. $\mathcal{M}$'s behaviour is summarised by:

$$
\begin{aligned}
b_i^{\mathcal{M}} &= 0 && (4.25)\\
\theta_i^{\mathcal{M}} &< w_{aut}\\
w_{read} &<< w_{aut}\\
\delta_u^{\mathcal{M}} &= 0 && (4.26)\\
\delta_v^{\mathcal{M}} &= \delta_{max}\\
S_i^{\mathcal{M}}(t) &= S_i^{\mathcal{B}}(t-1) = [v_i^{\mathcal{M}} > \theta_i^{\mathcal{M}}]\\
u_i^{\mathcal{M}}(t) &= w_{aut}\sum_{k=0}^{\tau_w}\rho(k)s_i^{\mathcal{B}}(t-k) + w_{read}\sum_{k=0}^{2}\rho(k)s_i^{\mathcal{T}}(t-k)\\
v_i^{\mathcal{M}}(t) &= u_i^{\mathcal{M}}(t) + \eta_i^{\mathcal{M}}(t).
\end{aligned}
$$

Axon Compartment $\mathcal{T}$. Because the compartmnts $\mathcal{B}$, $\mathcal{M}$, and $\mathcal{T}$ are updated sequentially, $\mathcal{T}$ is now able to read $S_i(t-1)$ and $u_i(t)$ from the stack and compute $v_i^{\mathcal{T}} = S_i^{\mathcal{M}}(t) \wedge u_i^{\mathcal{B}}(t) = \hat{\varepsilon}_i(t)$. In this ideal model, $S_i^{\mathcal{T}}$ signals satisfaction when no violations exist:

$$
S_i^{\mathcal{T}} = \begin{cases}
0, & (S_i^{\mathcal{M}} = 1) \wedge (u_i^{\mathcal{B}} < 0)) \;\rightarrow\; \text{violations exist}\\
1 & (S_i^{\mathcal{M}} = 0) \wedge (u_i^{\mathcal{B}} < 0)) \;\rightarrow\; \text{loser inhibited by competitors}\\
1 & (S_i^{\mathcal{M}} = 1) \wedge (u_i^{\mathcal{B}} = 0)) \;\rightarrow\; \text{winner with no violation}\\
1 & (S_i^{\mathcal{M}} = 0) \wedge (u_i^{\mathcal{B}} = 0)) \;\rightarrow\; \text{loser with no violation}
\end{cases} \tag{4.27}
$$

Hence, $\mathcal{T}$ should spike by default. For this, one could set $\theta_i^{\mathcal{T}} = 0$. However, the real equations on the chip are,

$$
\begin{aligned}
S_i^{\mathcal{T}}(t) &= [v_i^{\mathcal{T}} > \theta_i^{\mathcal{T}}], && (4.28)\\
v_i^{\mathcal{M}}(t) &= S_i^{\mathcal{M}}(t) \wedge (b_i^{\mathcal{T}} + u_i^{\mathcal{B}}(t)) + \eta_i^{\mathcal{T}}(t). && (4.29)
\end{aligned}
$$

Notice $\eta_i^{\mathcal{T}}(t)$ is always contributing independently of the spike and current variables from the stack. Hence, if $\theta_i^{\mathcal{T}} = 0$ noise will interfere with the neuron firing producing both false negatives and false positives. Then, we need to use $b_i^{\mathcal{T}} > |\eta| > 0$. Then,

equation 4.29 results in:

$$v_i^{\mathcal{T}}(t) = \begin{cases} b_i^{\mathcal{T}} + u_i^{\mathcal{B}} + \eta_i^{\mathcal{T}} < \theta_i^{\mathcal{T}}, & (S_i^{\mathcal{M}} = 1) \wedge (u_i^{\mathcal{B}} < 0)) \\ \eta_i^{\mathcal{T}} < \theta_i^{\mathcal{T}}, & (S_i^{\mathcal{M}} = 0) \wedge (u_i^{\mathcal{B}} < 0)) \\ b_i^{\mathcal{T}} + \eta_i^{\mathcal{T}} > \theta_i^{\mathcal{T}}, & (S_i^{\mathcal{M}} = 1) \wedge (u_i^{\mathcal{B}} = 0)) \\ \eta_i^{\mathcal{T}} < \theta_i^{\mathcal{T}}, & (S_i^{\mathcal{M}} = 0) \wedge (u_i^{\mathcal{B}} = 0)) \end{cases} \tag{4.30}$$

The red equations in 4.30 show that there is no signalling of satisfaction from $S_i^{\mathcal{T}}$ when $S_i^{\mathcal{M}} = 0$. This is because in equation 4.29 the spikes from $\mathcal{M}$ not only gate $u_i^{\mathcal{B}}(t)$ but also $b_i^{\mathcal{T}}$. Gating makes reference to the fact that the spikes state is binary and is multiplying the rest of the terms in the equation, if $S_i^{\mathcal{T}} = 0$, the current and bias are not integrated into $\mathcal{M}$'s voltage (Equation 4.29). We solve this problem by (see figure 4.11):

1. setting a non-zero bias current on the Σ neuron, label 4 in figure 4.11. Thus, Σ will fire by default.

2. create an inhibitory connection from $\mathcal{M}$ to Σ which is antisymmetric to that from $\mathcal{T}$ to Σ, red arrow in figure 4.11. Thus, each neuron will effectively inhibit Σ unless it has no conflicts, because in such a situation the neuron will send opposite signals of equal magnitude to Σ.

With such an strategy, the firing of Σ given the influence from the i-th neuron will obey:

$$s_{i \mapsto \Sigma} = \begin{cases} 1, & (s_i^{\mathcal{M}} = s_i^{\mathcal{T}} = 0) \wedge (u_i^{\mathcal{B}} = 0 \vee u_i^{\mathcal{B}} < 0) \\ 1 & (s_i^{\mathcal{M}} = s_i^{\mathcal{T}} = 1) \wedge (u_i^{\mathcal{B}} = 0) \\ 0 & (s_i^{\mathcal{M}} = s_i^{\mathcal{T}} = 1) \wedge (u_i^{\mathcal{B}} < 0) \end{cases} \tag{4.31}$$

For the red equation in 4.31, corresponding to the red conditions in equation 4.30, the *satisfied* condition is signaled by $b_\Sigma > \theta_\Sigma$ for either $u_i^{\mathcal{B}} = 0$ or $u_i^{\mathcal{B}} < 0$, because there is no input to Σ. In the second equation, the antisymmetric inhibition and excitation signals from neuron i compensate, having no net effect on u_Σ, labels 3 and 5 in figure 4.11. In the third equation, Σ is inhibited by $\mathcal{B}_i$ with no signal from $\mathcal{T}$, so it remains inhibited during the respective timestep acknowledging that a violation exists. This protocol solves the problem posed by equation 4.30.

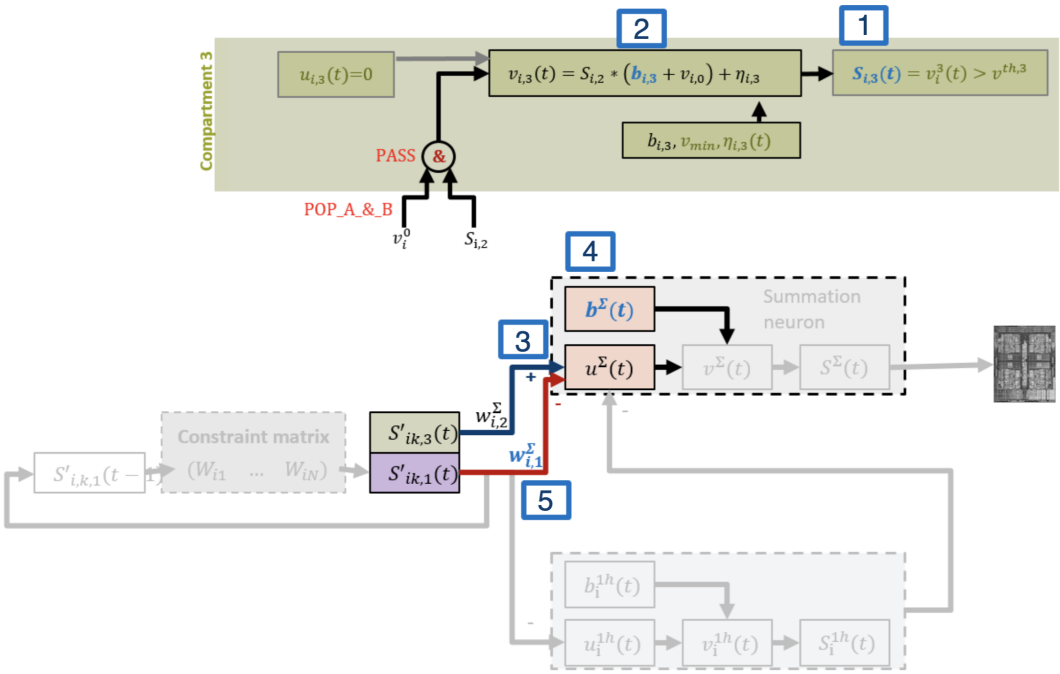


Figure 4.11: Protocol for signaling of satisfiability from a principal neuron onto the Σ neuron. An extra inhibitory synapse is created from the base compartment to the Σ neuron, which now has a non-zero bias current, see the text for description.

All in all, the dynamics of $\mathcal{T}$ can be summarised by:

$$
\begin{aligned}
b_i^{\mathcal{T}} &> |\eta| > 0 && (4.32)\\
b_i^{\mathcal{T}} &> \theta_i^{\mathcal{T}} \\
\delta^{u,\mathcal{T}} &= \delta_{max} \\
\delta^{v,\mathcal{T}} &= \delta_{max} \\
S_i^{\mathcal{T}}(t) &= [v_i^{\mathcal{T}} > \theta_i^{\mathcal{T}}] \\
u_i^{\mathcal{T}}(t) &= 0 \\
v_i^{\mathcal{M}}(t) &= S_i^{\mathcal{M}}(t) \wedge (b_i^{\mathcal{T}} + u_i^{\mathcal{B}}(t)) + \eta_i^{\mathcal{T}}(t) && (4.33)
\end{aligned}
$$

Both current and voltage decays are set to their maximum for this compartment so that it computes $\varepsilon_i(n\Delta t)$ for every timestep n. Note also that τ_w and $|\eta|$ constitute further free parameters which are shared by all compartments in the same neurocore.

4.3.4 Compensating for Noise

The model above satisfies the conditions for evaluating $\hat{\varepsilon}_i$ at every timestep. Nevertheless, because noise is defined by neurocore, it will be equally applied to every compartment. Thus, one needs to guarantee that noise will not cause interference on the auxiliary compartments by either producing spikes or cancelling them when they should occur.

Compartment $\mathcal{B}$

- $w_{ij} > |\eta|/2$, so that when $u_i^{\mathcal{B}}$ gets passed from $\mathcal{B}$ into $\mathcal{T}$, it causes inhibition for all $u_i^{\mathcal{B}} < 0$.

- Optionally, if noise should not cause spikes, $\theta_i^{\mathcal{B}} > |\eta|/2$. Note that if $|\eta|/2 \approx \theta_i^{\mathcal{B}}$ the inter-spike interval will not follow $\theta_i^{\mathcal{B}}/b_i^{\mathcal{B}}$. Also, if $|\eta|/2 >> \theta_i^{\mathcal{B}}$ the compartment will tend to spike at every timestep which is not desirable.

Compartment $\mathcal{M}$

- For this compartment one should guarantee spike-in-spike-out, i.e. $S_2(t) = S_1(t-1)$. Hence $|\eta|/2 < \theta_2 < w_{12} - |\eta|/2$.

Compartment $\mathcal{T}$

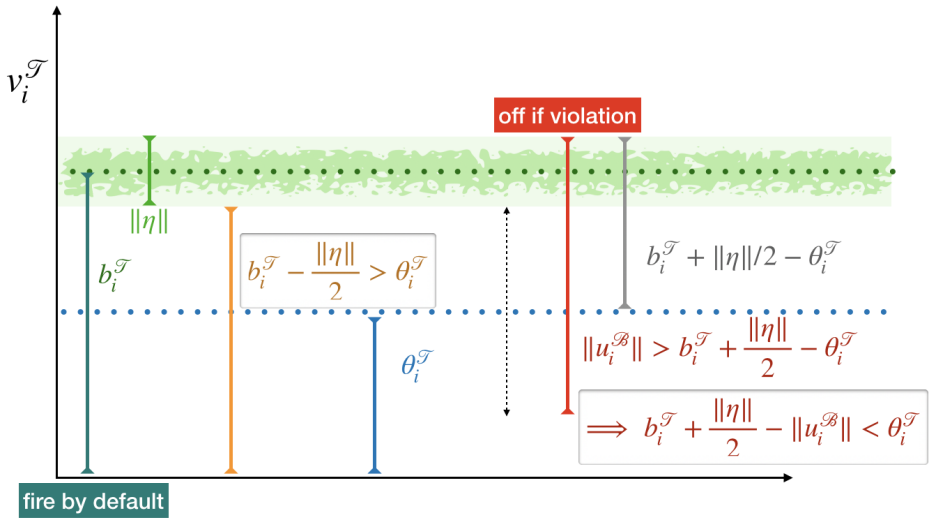


Figure 4.12: Contributions to compartment voltage for $\mathcal{T}$ showing restrictions for $\theta_i^{\mathcal{T}}$. The dashed black double headed arrow shows the range of possible values for $\theta_i^{\mathcal{T}}$ which satisfy both conditions highlighted with a shadowing box. Note there is no particular meaning for the horizontal axis.

We show in figure 4.12 the contributions to the compartment voltage of $\mathcal{T}$, which will integrate both its bias current (dotted green line), the input current from compartment $\mathcal{B}$ (red arrow) and the LFSR noise, represented as the shadows around $b_i^{\mathcal{T}}$ in the figure.

- Every compartment $\mathcal{T}$ should fire by default and noise should not interfere with firing, orange condition in figure 4.12. Thus, $\theta_i^{\mathcal{T}} < b_i^{\mathcal{T}} - |\eta|/2$.

- Input spikes to the multicompartment neuron should make the compartment $\mathcal{T}$ to stop firing. In figure 4.12, this means that when the red arrow is substract from the maximum value $b_i^{\mathcal{T}} + |\eta|/2$ the voltage should result in a value below the blue dotted line (threshold). Then, $|u_i^{\mathcal{B}}| > b_i^{\mathcal{T}} + |\eta|/2 - \theta_i^{\mathcal{T}}$

- Both conditions above imply:

$$b_i^{\mathcal{T}} + \frac{|\eta|}{2} - |u_i^{\mathcal{B}}| < \theta_i^{\mathcal{T}} < 2b_i^{\mathcal{T}} - \frac{|\eta|}{2} \tag{4.34}$$

  which defines the region marked with a dotted double headed arrow in figure 4.12,

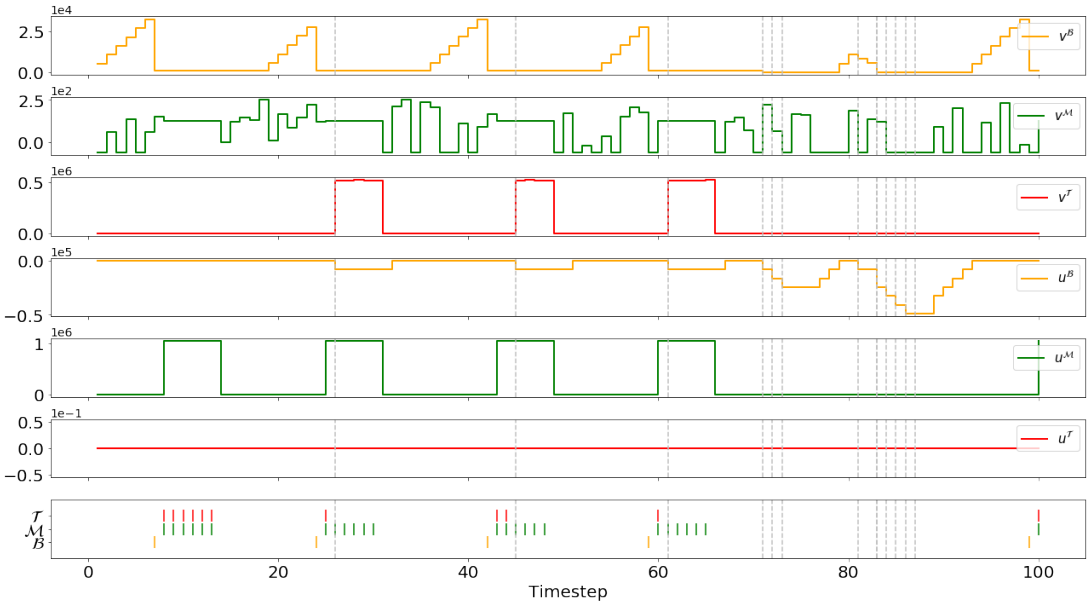


Figure 4.13: Activity of the ($\mathcal{B}$,$\mathcal{M}$,$\mathcal{T}$) multicompartment neuron. The top three pannels show the compartments voltage and panels four to six the compartments current for $\mathcal{B}$,$\mathcal{M}$ and $\mathcal{T}$ respectivelly. The spiking activity is plotted on the bottom panel. Grey dashed vertical lines represent the times of injection of inhibitory spikes.

we chose the halfway, resulting in:

$$\theta_i^{\mathcal{T}} = \frac{2b_i^{\mathcal{T}} - |u_i^{\mathcal{B}}|}{2}. \tag{4.35}$$

4.3.5 3-Compartment Model Implementation

Fig 4.13 shows the measured behaviour of the three state variables, $v(t)$, $u(t)$ and $s(t)$ for compartments $\mathcal{B}$, $\mathcal{M}$ and $\mathcal{T}$ in a single 3-multicompartment neuron implemented in Loihi. These base, middle and top compartments are colour-coded as yellow, green and red, respectively. As explained above, the noise has been enabled for $v(t)$ (evident for $v^{\mathcal{M}}$, in green), and we have asserted the established parameter ranges so that η does not interfere with the desired functionality. In order to test the influence of presynaptic neurons in a controlled manner, we have injected external inhibitory box spikes whose start is shown as vertical dashed grey lines. The firing times of the injected spikes were [25, 44, 60, 70, 71, 72, 80, 82, 82, 83, 84, 85, 86] and are intended to test the model works as expected. In the top panel, $v^{\mathcal{B}}$ accumulates bias until the first yellow spike appears on the lowest panel. $v^{\mathcal{B}}$ resets after spiking for the duration of the refractory period and repeats the process over again. The spiking activity of the three compartments

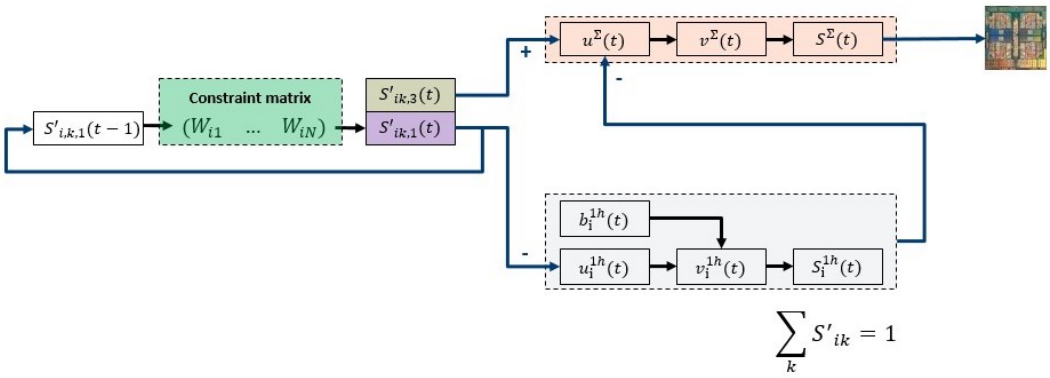


Figure 4.14: Schematic representation of the on-chip neural architecture for network energy integration. The principal network is build from the base compartments (purple) with their connectivity defined by a weighted adjacency matrix W (green). The top compartments (dark green) send local (binary) information to the summation neuron Σ (rose) about their conflict state with the rest of the network. An extra neuron ζ (grey) is needed to notify Σ about false positives when no-neuron from a given WTA is firing. When no conflicts exist on the network, Σ notifies the LMT CPU (rigth-top).

shows that every $\mathcal{B}$ spike (yellow) triggers a spike on $\mathcal{M}$ (green) on the next timestep. Before the first presynaptic spike arrives ($t = 25$), $\mathcal{T}$ spikes simultaneously to $\mathcal{B}$ because no conflicts exist and their updating belong to the same timestep. However, incoming inhibitory input turns off the firing from $\mathcal{T}$ at time 25 when the first spike arrives. Note in the fourth panel (in yellow) that $u^{\mathcal{B}}$ integrates such spikes as box synapses, these accumulate in $v^{\mathcal{T}}$ when it is not refractory. Because $\mathcal{T}$ resets at $t+1$ if it spiked at t, its $v^{\mathcal{T}}$ plot does not capture such changes. However, when $\mathcal{T}$ does not fire, because integrates $u^{\mathcal{B}} \neq 0$. a square pulse appears on $v^{\mathcal{T}}$, reflecting $b^{\mathcal{B}} - u^{\mathcal{B}} < \theta^{\mathcal{T}}$. Presynaptic spikes, do not affect the firing of $\mathcal{M}$ as its spiking is necessary for the functioning of $\mathcal{T}$. After timestep 70, when there is enough inhibition to turn off the neuron, none of the compartments spike.

4.3.6 Σ and ζ Neuron Implementation Details

Because $\mathcal{T}$ computes $v^{\mathcal{T}}$ from equation 4.29, the configuration of Σ's current bias and threshold should make it fire constantly, but be inhibited at the arrival of any signal of violation from any $\mathcal{M}_i$ compartment. Recall Σ is excited by $\mathcal{T}$ with weight $w_{\mathcal{T},\Sigma}$ and inhibited by $\mathcal{M}$ with an exact and opposite weight $w_{\mathcal{M},\Sigma} = -w_{\mathcal{T},\Sigma}$, see figure 4.11. If both axonic compartments fire, is because the neuron has no conflicts, then, Σ is not

effectively inhibited by this neuron. If on the contrary only the second compartment spikes, it is because the neuron received an inhibitory input on $\mathcal{B}_i$, a conflict exists, $u_i^{\mathcal{T}}$ turns off $\mathcal{T}$, and the second compartment effectively inhibits Σ.

When trying to solve large CSPs, it became evident that, although we are showing a single summation neuron here, this cannot always be the case. For large CSPs where a single Σ neuron would need to integrate information from more neurons than the fan-in axons allowed for a single neurocore, a hierarchy of Σ neurons has to be used as follows. The total population of WTAs is partitioned to create N_σ subpopulations whose size fits the number-of-axons-per-core constraint, the first layer of Σ neurons will integrate satisfiability from all these subpopulations of WTAs. The Σ neurons then feed-forward to a single, and slightly different, integration neuron I, which acts as the global arbiter. I should have $\theta^I = N_\Sigma / w_{\Sigma,I}$, where $w_{\Sigma,I}$ is the weight of each synapse from Σ_i to I (the same for all).

This hierarchical construction can be used in any general non-CSP SNNs where one needs to monitor specific subpopulations which should satisfy certain consistency constraints. For example, it has been demonstrated experimentally that the brain encodes memories in ensembles of neurons where a few neurons are active, these ensembles are used to do elaborate computations or create memory associations [PIR+19, Qui16]. In a system like this, several summation neurons will be spread across the network to monitor specific subsets of neurons, notifying other SNNs or SNIPs when the underlying conditions have been fulfilled. In such a general SNN, not all neurons would need to be 3-multicompartment neurons as that of figure 4.10, only those subsets whose state, or $\hat{\varepsilon}$, hold relevant information. In general, the architecture demonstrated here can be integrated into any network where its functionality renders useful.

Back into the case of a single Σ neuron, because $u_\Sigma = \sum_i w_{i\Sigma} s_i^{\mathcal{T}}$, where $w_{i\Sigma}$ is the weight of the excitatory connection between principal neurons i and the Σ neuron. Then, $\theta_\Sigma = N_\sigma w_{i\Sigma}$, so that this neuron fires when all principal neurons acknowledge a non-conflicting state. This protocol generates a coarsely quantised energy function because spikes can only transfer binary variables, unless coding through their timing. Each incoming spike from a neuron without conflicts will contribute the same energy to u_Σ independently of the number of violations it experiences. Still, the temporal dependence of u_Σ will reflect a proxy $\hat{E}$ of E. In particular, one is interested in the extrema of E or states with a particular value of it, if the target is a value $\mathbf{S}(E \neq 0)$, one could modify θ_Σ so that Σ will fire when the network reaches the desired energy level. Note that the coarseness of $\hat{E}$ worsens as more Σ layers are added. But still, $E = 0 \iff \hat{E} = 0$, so

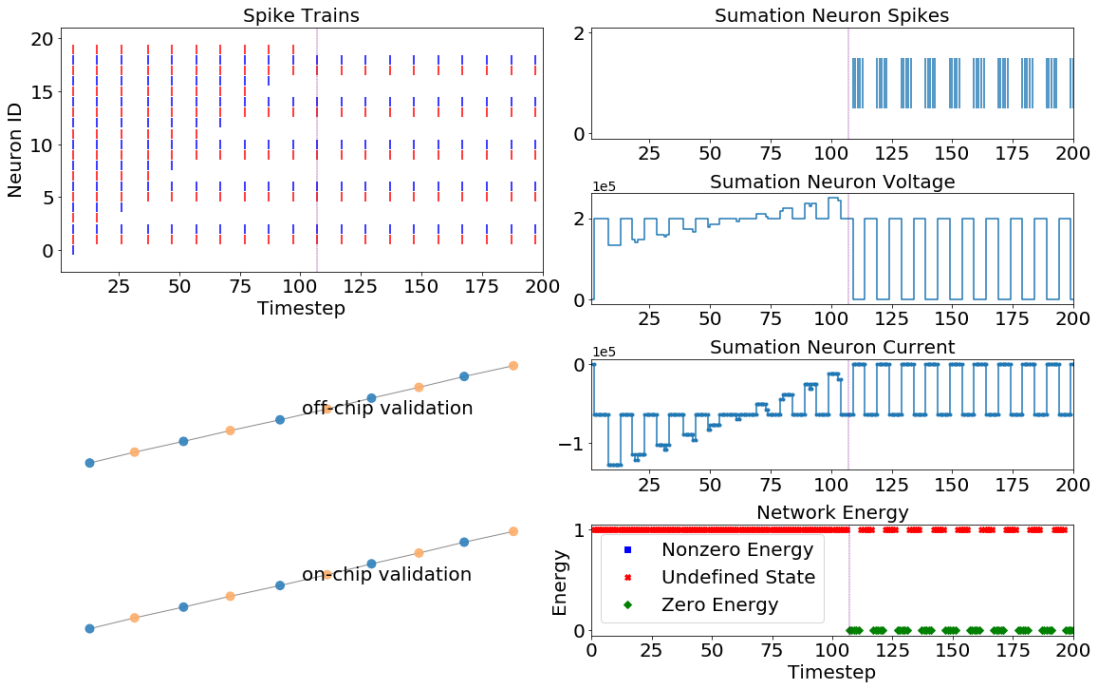


Figure 4.15: On-chip vs off-chip evaluation of E for a small spin chain. Excitatory input has been used to control the progressive relaxation to the ground state. Notice the resemblance of the summation current as a proxy for E shown in the right bottom plot. As soon as $E = 0$, $u_\Sigma = 0$ and the summation neuron begins to fire, except for the intervals where the network state is undefined. Vertical red and blue lines (superimposed) represent the offline and online times to solution respectively.

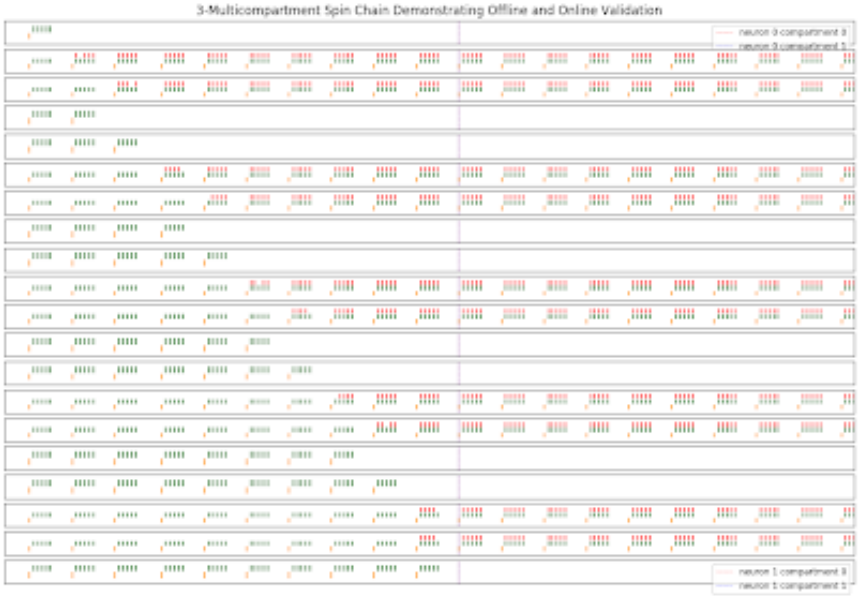


Figure 4.16: Spiking activity of all 3-multicompartment neurons in the spin chain of figure 4.15, one per panel. Yellow, green and red event lines represent the spike times of base, middle and top compartments of a multicompartment neuron respectively. Every two neurons represent a spin variable with up and down states. The long vertical purple line represents the superimposed offline and online times to solution respectively

the CSP functionality is never lost.

One should only measure E when valid states of the ensembles exist, avoiding false positives when the network dynamics gives rise to under-determined states. A false positive will occur if Σ measures $E = 0$ because certain WTAs were not active at all and the only active WTAs did not have violations. Inactive WTAs have energy zero but represent undefined variables, then, a partial assignment of variables that satisfies all constraints has energy 0 but does not solve the CSP problem. Still, the problem will not be solved, and the information encoded in the network state is not useful. Hence, an auxiliary state-enforcement neuron ζ (grey box in figure 4.14) is set for each WTA to inhibit Σ if the respective WTA is idle. Every ζ neuron is set to fire by default, the $\mathcal{B}$ compartments of the respective WTA, in turn, inhibit the ζ neuron. This means ζ does not fire whenever one or more WTA neurons fire which implies the population has a defined state or is actively searching, in which case conflicts will be acknowledged to Σ directly by the WTA neurons. The state from $\mathcal{B}$ takes two timesteps to arrive at Σ

because of the one timestep delay between $\mathcal{B}$ and $\mathcal{M}$. The same delay applies for the propagation of information from $\mathcal{B}$ to ζ to Σ, so the system remains coherent. We say the layer of ζ neurons enforces the one-hot representation,

$$S_{i\cdot|D_i|+k} = S'_{ik} \mid S'_{ik} \in \{0,1\} \wedge \sum_k S'_{ik} = 1. \tag{4.36}$$

It seems that the all-to-one connectivity will generate substantial spike traffic. Recall, however, that spikes are only sent to Σ whenever neurons are in the satisfying configurations. Thus, the firing is expected to be sparse most of the time, until complete satisfaction, when the neurocore finishes its computation.

4.3.7 Multicompartment WTAs

Now, we turn to test the model works when using multiple neurons.

4.3.7.1 One Multicompartment WTA

In figure 4.17, the top 3 panels show an ensemble of three 3-multicompartment neurons whose dynamics correspond to that of figure 4.13. The neurons inhibit each other laterally so that they encode a random variable in a one-hot representation. The bottom three panels illustrate the activity of the state-enforcement neuron (grey block in figure 4.14), which stops firing whenever at least one neuron in the ensemble is firing. Notice that due to the lateral inhibition, the third compartment (red spikes) does not fire during the periods where more than one neuron in the ensemble is active, which causes $E \neq 0$.

4.3.7.2 Network of WTAs with Σ and ζ Neurons

To illustrate the activity of the whole neural architecture, we built two neural ensembles similar to that of figure 4.17 with the condition (cross-inhibition) that both random variables should not assume the same value, the spiking activity of the six multicompartments is shown by the top 6 panels of figure 4.18. The lower panels show the activity of the Σ and ζ neurons (pink and grey blocks in figure 4.14). Clearly, the summation neuron fires (third panel from bottom-up) only when the network is in its ground state and both populations acknowledge their no-conflicts state.

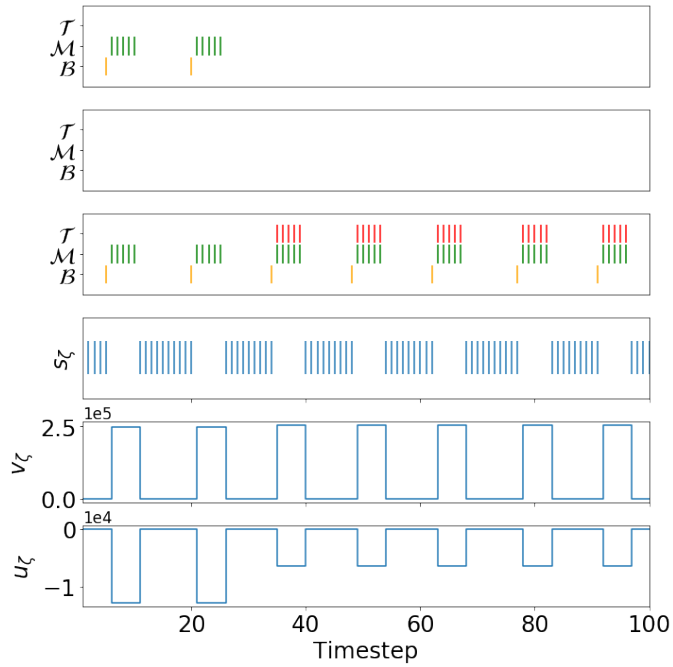


Figure 4.17: ζ neuron (bottom three panels) with a multicompartment WTA made of three neurons (top three panels). Notice how ζ fires whenever no neuron is active and stops when at least one neuron is firing.

4.3.8 Solution Readout

To this point, each neuron computes $\hat{\varepsilon}_i$, send it to Σ. Σ integrates $\hat{E}$ with help from ζ to detect false positives and fires if $\hat{E} = E = 0$. The next steps consist of detecting s_Σ with an LMT, so that the host knows when to read out the solving network state.

4.3.8.1 Sequential Neural Interfacing Processes

We can create communication channels between the Σ neuron and any LMT processor (or any other neuron), as well as from that LMT and the superhost. We create a process whereby if Σ fires, it will trigger the following execution (4.21):

1. LMT sends the current time to the superhost
2. Superhost requests the neurocores for the current network state of every principal neuron.
3. LMT reads the state variables of interest from the principal population
4. The state variables are transferred to the superhost.

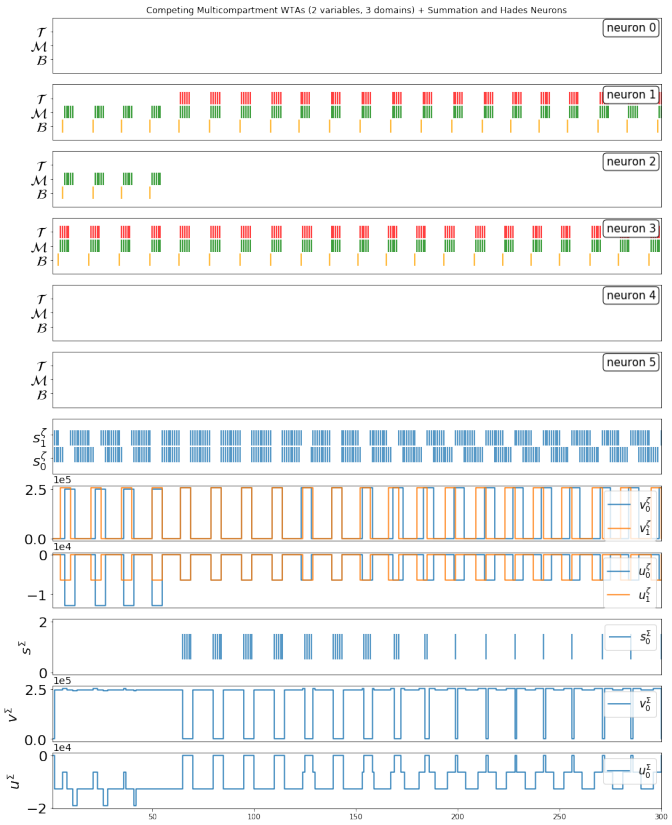


Figure 4.18: Σ neuron (bottom three panels) for a network with two WTAs (top six panels) as that of figure 4.17. There is one ζ neuron per WTA. Note Σ never fires if ζ is active, also, Σ fires only when both neurons have converged to a winner which is compatible with the non-equal constraint.

5. Host decodes the original network state

Decoding is necessary because of a three timesteps delay for information to flow from the principal neurons to the LMTs. Once the network solves the CSP, at t_s, the $\mathcal{T}$ compartment of the last neuron which generated t_s will spike at $t_s + 1$. At $t_s + 2$, Σ will recognize that the network solved the problem and will, in turn, send a message to the LMT, which arrives at $t + 3$. However, the network state could have already changed at $t_s + 3$. For this reason, we need to store the network state during this small, but important, delay.

Our original method for readout consists of using decaying activity traces, originally intended for homeostasis (figure 4.19). Such traces can be activated for any compartment, with characteristic gain and decay. The decay of the original traces includes stochastic rounding, making its use to store $s_{\mathcal{B}}$ unreliable. We used NxCore to make the decay deterministic so that the value of each trace reflects the neuron state from 3

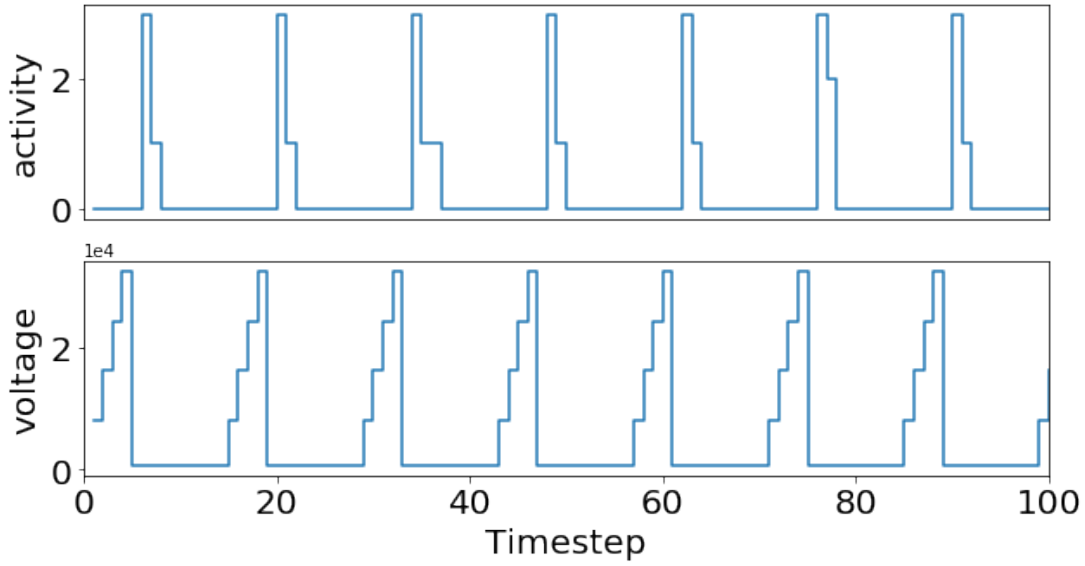


Figure 4.19: Soma activity traces with default stochastic rounding for decay (top). Dashed grey lines indicate the spike train triggered by a bias current which accumulates on the compartment voltage (bottom).

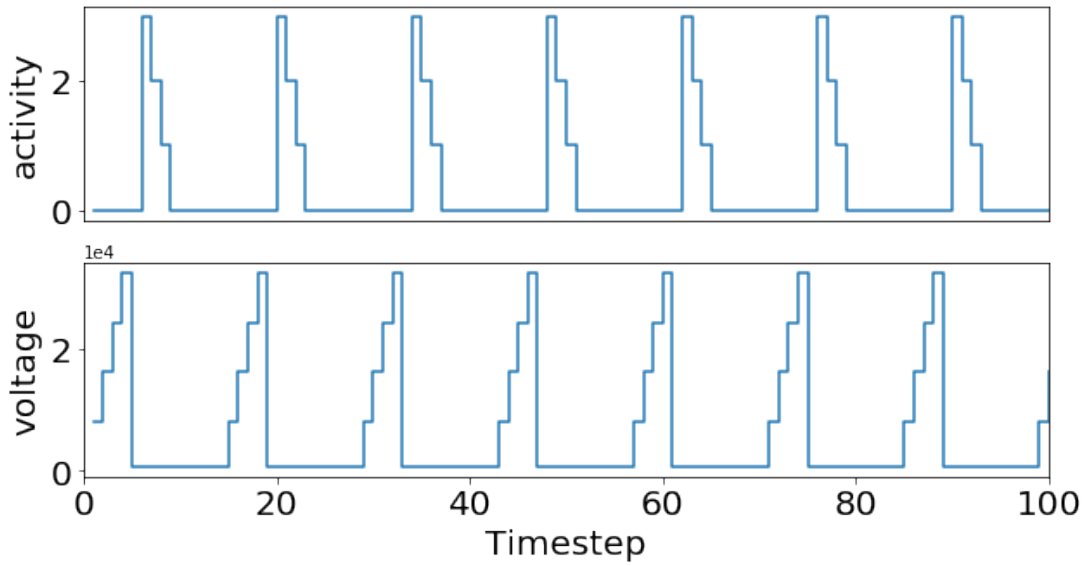


Figure 4.20: Soma activity traces without default stochastic rounding can be used to read out network state after a given number of timesteps.

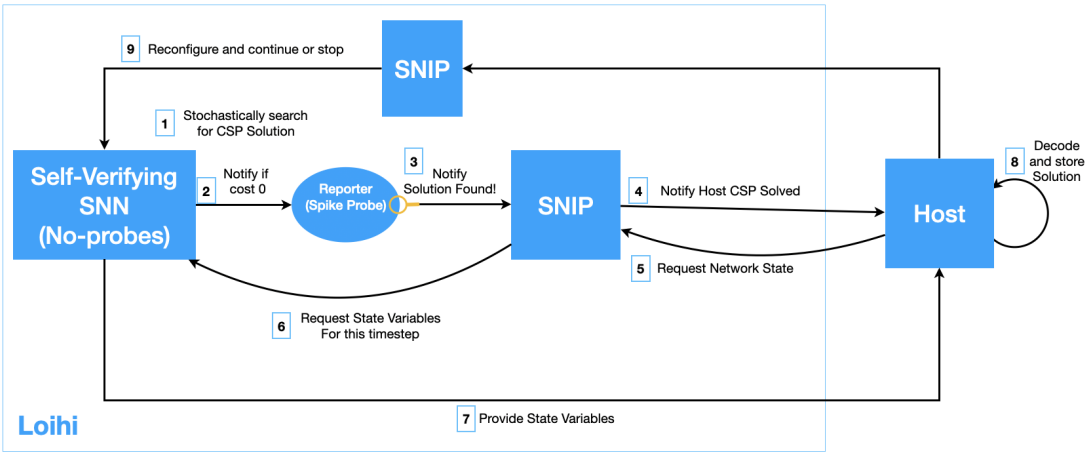


Figure 4.21: Computational graph for the network state readout SNIPs. Time unfolds to the right.

timesteps into the past.

Using activity traces is economical and our preferred method. However, given some unexpected behaviour, we had to use an alternative method. It consists in creating an extra autapse from $\mathcal{T}$ to $\mathcal{M}$ with a negligible excitatory weight $w_{read} << w_{aut}$, so that it causes no interference. When $\mathcal{T}$ spikes, it sends a box autapse with $\tau_w = 2$, so that $v^{\mathcal{M}}$ stores $s^{\mathcal{T}}$ during two timesteps. When the Host gets notified, it reads all $v^{\mathcal{M}_i S}$ and detects the small fingerprint. In this way, the Host reads the network state that solved the CSP.

To complete the implementation of the on-chip validation solver, we create python SNIPs to instruct the Host and a C SNIP to be run by the LMT.

4.3.8.2 SNIPS Implementation in Python

Listing 4.1 shows the python SNIP for readout. It consists of a *setup_snips* function, whose input is a pre-compiled loihi board. The function defines the C SNIP, chip ID and core ID to run. *create_notification_channels* and *_connect_sumation_neuron_to_lmt* (listing 4.2) create the necessary communication channels across the heterogeneous hardware stack. Finally, a helper function tells the system whether to use activity traces or readout autapse.

Table 4.4: Additional constructor attributes for the CSPNxNet API with on-chip validation.

MULTICOMPARTMENT	ON-CHIP VALIDATION	CHIP
multicompartment	set_summation	num_dendritic_acumulators
noise_at_multicompartment	set_one_hot_enforcement	num_delay_bits
enable_activity_traces	setup_snips	neurons_per_core
enable_refractory	readout_autapse_delay	
	online_window	
	multiple_summation	

4.3.8.3 SNIPS Implementation in C

On the C side, a guard function (Listing 4.3) controls the execution of the main function (Listing 4.4). This latter, in turn, detects the Σ spiking and transfers the firing timestep to the host.

4.3.9 Update of the API Design for Multi-compartment on-chip Validation

The extra functions for on-chip validation explained through the previous sections, have been integrated into the CSPNxNet API. Table 4.4 shows the on-chip validation solver attributes available through the constructor, table 4.1 shows the properties concerned with multicompartments in its second column. In figure 4.3, the methods in the *ONLINE* block set up most of the on-chip validation architecture. Nevertheless, the *PROTOTYPES*, *PROBES*, *BUILD NET* and *DRAWING* blocks had to be augmented to support the online validation and multicompartment features of the system.

4.3.10 Online Solution and Validation of CSPs

Figure 4.22 shows the off-chip and on-chip validation methodologies for finding the solution to the 3-colouring of the map of Australia. On the right, the red and blue vertical lines (superimposed) mark the time to solution, at which Σ begins firing acknowledging $E = 0$. This time is consistent across the CSN raster plot (top), summation activity (spikes, voltage and current on the three central panels), and off-chip computation of E, bottom panel. The constraint graph labelled as *on-chip evaluation* corresponds to the solution retrieved online without probing any compartment. Note that u_Σ serves as a proxy to track the evolution of the stochastic search. This is evident in figures 4.15 and

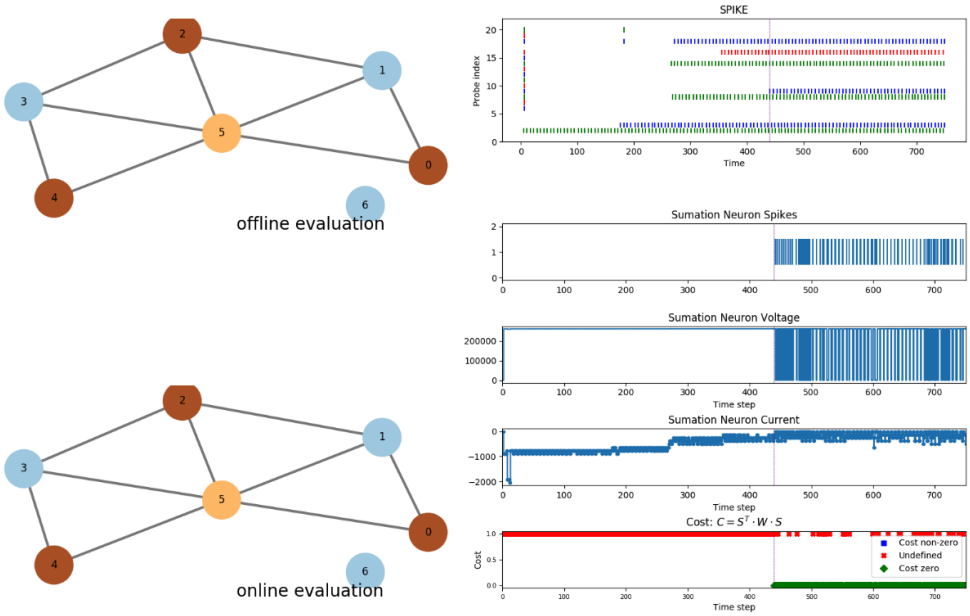


Figure 4.22: Comparison of on-chip and off-chip validation of solution to the 3-colouring map of Australia. The constraint networks to the left show the colouring as retrieved with off-chip validation (top) and with on-chip validation and no probes(bottom). The plots to the right show the probed activity for spikes from principal neurons (top), summation neuron spikes, voltage and current (middle blue), and off-chip evaluation of E (bottom). Red and blue vertical lines (superimposed) label the time to solution from off-chip and on-chip validation respectively.

4.16 where we have forced the progressive relaxation to the ground state for a small spin chain. Note the direct correspondence between u_Σ and E.

4.3.10.1 Building Apps

The CSPNxNet API is intended for creating applications for specific CSPs. These should hide most of the low-level details to the end user. To build these apps one generally implements a translator to have the problem readily encoded as input. Then, the user should do a minimal call to the solver specifying the input and validation mode:

```
import solver
net=solver(CSP instance, mode="online")
net.run()
net.plot()
```

We have implemented such applications for both colouring map problems and Latin squares (including Sudoku) (see section 2.6.2). Calling the latter for sizes from 1 to 12 reproduces the figure 4.23. In which all puzzles have been solved with on-chip

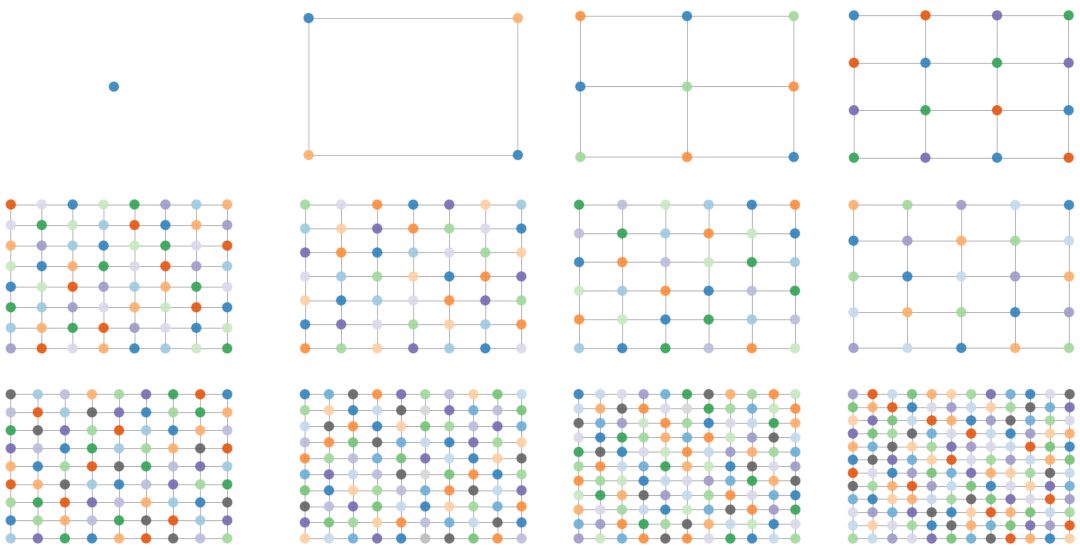


Figure 4.23: On-chip solution to Latin squares of increasing sizes, the same digit cannot appear more than once per row and per column, digits are encode by colours to ease visualisation.

validation. Figure 4.24 shows the time per timestep for each one of these problems. The mean solving time and energy consumption are shown in figure 4.25.

Note, however, that execution time per timestep is expected to be in the range of 10-50 μs, while figure 4.24 shows timesteps of 400 μs, suggesting the existence of a software or hardware bug. Nevertheless, further investigation is needed to confirm these facts and, if necessary, perform optimisations. Still, by verifying the solution on-chip, the total overhead of extracting and postprocessing data, which is in the order of seconds, has been eliminated. The API has not been optimised fully, so we foresee further improvements to exploit the capabilities of the chip.

4.3.10.2 Buffer Layer to Speed Up Solution Detection

We have demonstrated the functioning of our strategy for on-chip validation of CSP solutions. Such a protocol, however, shares a weakness with the off-chip validation, namely, the possibility to miss a solution when the firing of the neurons encoding such a solution does not co-exist in the time window used to measure E. When validating on the host, this means one should sweep the box length when determining if the network found a solution or not. This problem becomes more critical as the problem size increases for two reasons. Firstly, the probability of randomly firing neurons to coincide in a given

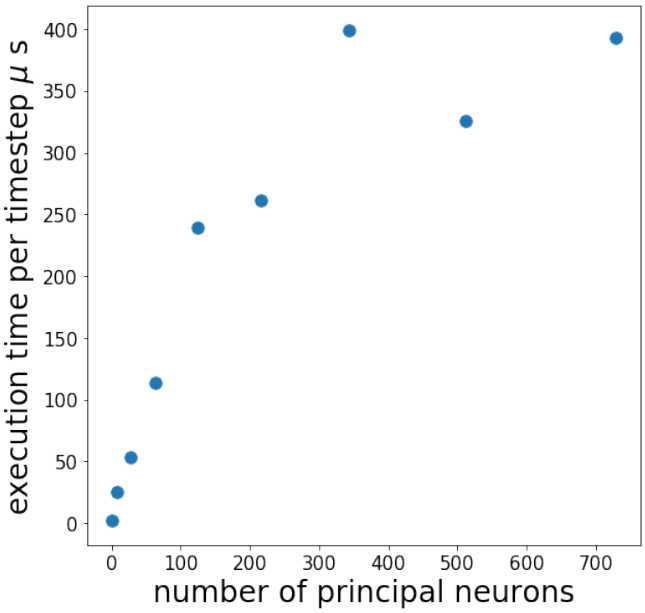


Figure 4.24: Time per timestep vs problem size for the Latin squares in figure 4.23.

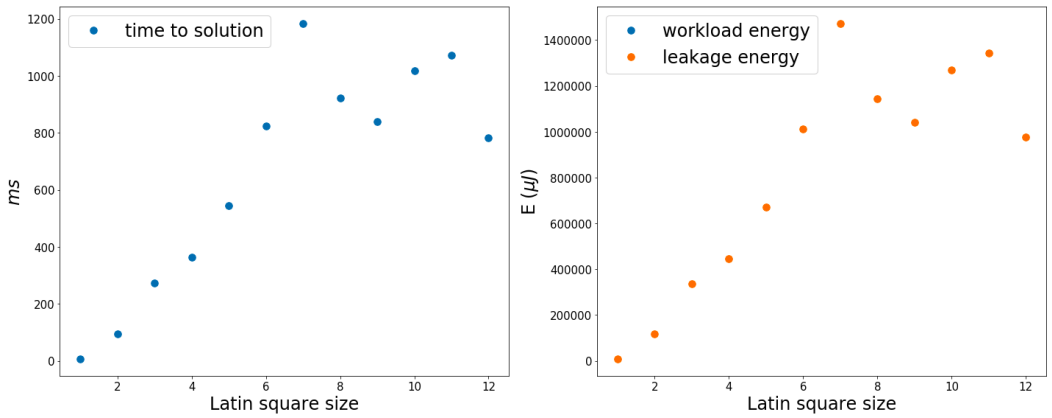


Figure 4.25: Execution time (left) and consumption energy (right) vs puzzle size for the Latin squares in figure 4.23.

time window decreases with the number of neurons to be considered. Secondly, more extensive networks require a larger θ_i/b_i ratio, which governs the average inter-spike interval, to allow for stochasticity, as well as to avoid deadlock by synchronisation. This latter occurs when two neurons fire at the same time, which would, in principle help with the coincidence in the readout time window. However, recall the symmetry on the connections of our networks. These reflect the equiprobability that should be assigned to the possible valuations of competing variables, but when simultaneous spiking happens in symmetrically connected neurons, the inhibition will not result in any competition. We prevent such spurious synchronisation by both randomising the initial voltage of all neurons to different values, as well as keeping θ_i/b_i large enough so that the LFSR noise reflects in stochastic behaviour of the voltage increase in every inter-spike interval. Hence, it is expected that when large networks converge to the equilibrium firing distribution, the spikes are sparse in time through intervals that can potentially reach hundreds of timesteps. Nevertheless, the maximum PSP lenght for box synapses in Loihi is eight timesteps when using 1024 neurons per chip. For every new bit we use for the box synapse, we sacrifice half of the compartments that can be used in a core, a situation that renders the use of larger box lengths unfeasible, actually impossible for more than 10 bits. In the following, we present the design of a readout layer of neurons which buffers the last valid state of the network allowing for fast detection of a solution with arbitrary sparsity.

Figure 4.26 shows the architecture and activity of the proposed buffer layer. This example shows the functionality for storing the last value of a WTA population representing a CSP variable with five possible values encoded by colours in figure 4.26. The spikes, voltage and current plots correspond to the neurons in the buffer layer with colours encoding the neuron which is presynaptic to each buffer neuron. The firing of the principal neurons is shown colour-coded as vertical dashed lines in the spikes plot of the buffer layer. We want the buffer layer to keep firing the neuron which is postsynaptic to the last active principal neuron independently of its time sparsity. The blue neuron spikes first with spiking times [0, 1, 1, 12, 15]. The orange neuron then begins spiking at time step 23. Notice that the blue buffer neuron spikes regularly from time step 0 up to time step 23 when it stops firing, and the orange neuron takes on. The same behaviour is consistent across the firing of the subsequent neurons. In order to ease visualisation, the neurons are activated sequentially with random firing and allocated intervals of 20 timesteps for each one. To achieve the switching of the active neuron for each change of valuation, we require for an incoming spike to activate the respective

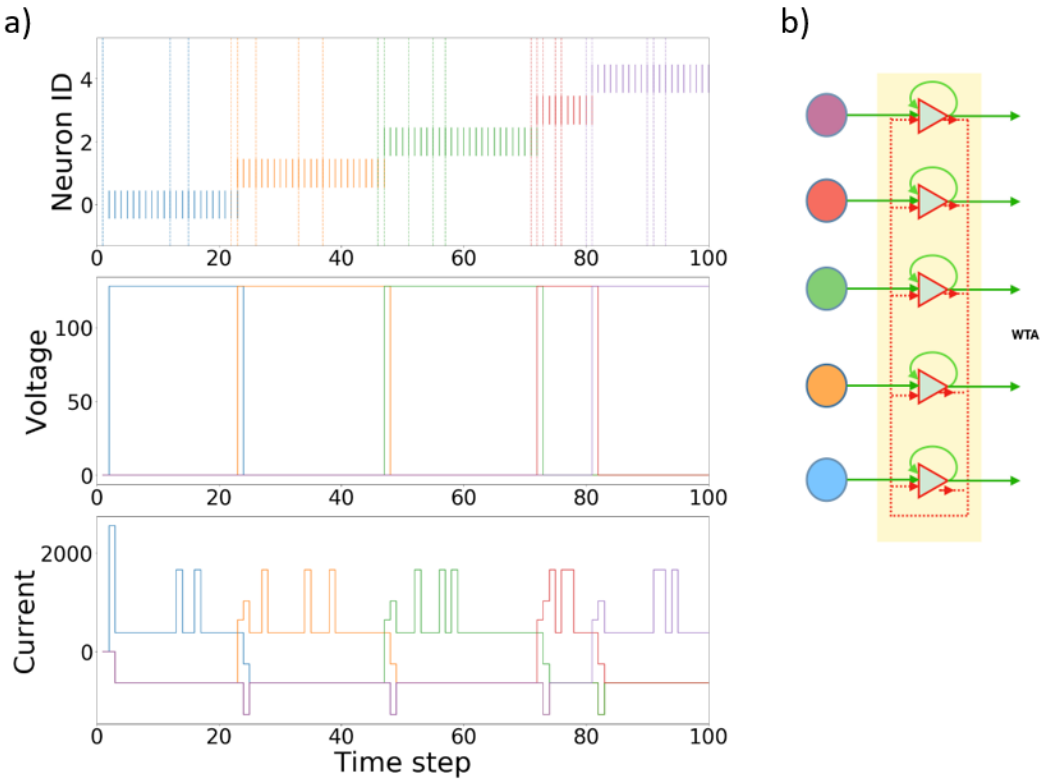


Figure 4.26: Implementation of a neural buffer layer which stores the last valid state of an SNN. Here, we represent the activity for a single WTA of our CSP network. The architecture is shown in b) for five principal neurons whose identity is encoded by colours. The same colour code is used in the plots in a) for the spikes, voltage and current of the corresponding buffer neuron (grey triangular neurons). The principal neurons feedforward excitation to the buffer layer on the yellow square. Dashed vertical lines in the raster plot show the spikes of every principal neuron. Notice that the activity of the buffer layer encodes the last active neuron.

postsynaptic neuron and deactivate its set complement, the neuron should remain active until other principal neuron spikes, which in turn deactivates it and turns on the neuron encoding the new winner. Activity is kept on by assigning self-excitation to each buffer neuron, switching off the complementary neurons is done by lateral inhibition between all buffer neurons mapping to the same WTA. The constraints for such architecture to work are:

$$\delta_u \mapsto \infty \tag{4.37}$$

$$\delta_v \mapsto \infty \tag{4.38}$$

$$0 < \theta_b < w_{self} < w_{inh} < w_{inh} + w_{self} < w_{triger} \tag{4.39}$$

$$w_{triger} + w_{self} > v_{th} + 2 * w_{inh}. \tag{4.40}$$

Where w_{triger} is the weight for synapses from principal neurons to the buffer layer, w_{self} is the weight for self exccitation (green loops in figure 4.26), which allows the buffer neurons to sustain activity. w_{inh} is the weight of latteral inhibitory connections between buffer neurons and θ_b is their voltage threshold.

4.4 Conclusion

In this chapter, we demonstrated the development of a solver for constraint satisfaction problems with both on-chip and off-chip validation. We have demonstrated a neuron design which harnesses the multicompartment operations and the hierarchical hardware structure offered by the Loihi systems architecture, to obtain a scalar measure analogous to the energy of a physical system. Tracking the activity of the network in this way eliminates the need for probing the state variables of each neuron in order to know the network activity or trigger other processes in the network. By modifying the biases and thresholds of the axonic compartments, it is possible to set target values for the proxy of the energy function at which the network stops the simulation and informs the micro-state of the system. This, in turn, can be used to trigger network modifications, sample the network and continue the run, or to stop the simulation altogether for scenarios where a decision or the result of a computation can be decoded from the output network state.

Here we integrate the energy across the whole network. It may also be useful to implement this architecture sparsely in certain SNNs as a neural motif for feedback

control.

The mathematical operations being carried out by the network are almost the same as those for post-processing. Thus, despite the requirement for additional neural and synaptic resources, we are improving performance by avoiding unnecessary data storage and events traffic across heterogeneous computational hardware.

Part II

Neuronal Excitability from Cellular-level Stochastic Dynamics

Chapter 5

Neuromorphic Implementation of Postsynaptic Currents

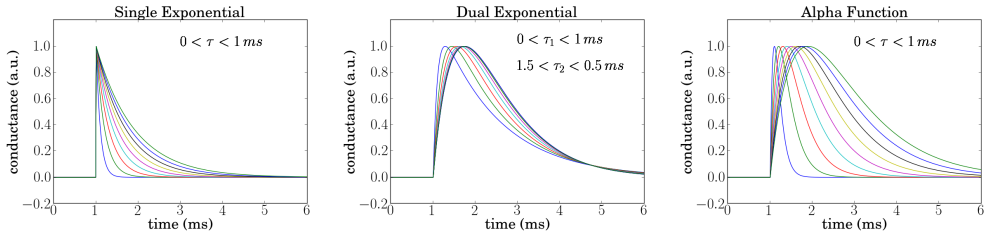
In this chapter, I propose a methodology for the implementation of the alpha, dual exponential and rectangular postsynaptic currents (PSCs) into the Spiking Neural Network Architecture (SpiNNaker). These currents are widely used in both theoretical developments [PIR+19, Flo12] and analysis of experimental results [LSH+02, HR97]. The characteristic flexibility of SpiNNaker allows the unique ability of totally modifying neural models and features. This and the next chapter advance the biological detail of the neural models in SpiNNaker so that more experimental data can be integrated with neuromorphic simulations, resulting in testing and improving models and hypothesis of neural dynamics in behaviour and high-level cognitive tasks.

Consider an arbitrary neuron l in a network of spiking neurons. If a second neuron μ is presynaptic to l and emits a spike at a time $t_m^{(f)}$, it will generate a postsynaptic current $j(t-t_m^{(f)})$ into the neuron l. In a general case, every single neuron receives spikes from several presynaptic cells. Thus, at an arbitrary time t, the total current induced in neuron l is given by:

$$J_l(t) = \sum_m w_{lm} \sum_f j(t-t_m^{(f)}), \tag{5.1}$$

where the subindex m runs over every neuron that is presynaptic to l and w_{lm} is the weight of each synapse from neuron m to neuron l. The index f accounts for multiple firing times from the same presynaptic neuron.

Several expressions, having a different level of realism, are available to represent j, the most commonly used functions in implementations of networks of spiking neurons



(a) single exponential (b) dual exponential (c) alpha function

Figure 5.1: Standard types of postsynaptic conductance changes showing their dependence with the decaying time constants.

are the exponential, dual exponential and alpha functions. However, only the former was implemented on the sPyNNaker module [RBB+18]. This is because the other PSCs were thought to be non-linearizable, meaning that memory requirements would grow with the number of synaptic events, making their implementation in SpiNNaker impracticable. This document aims to provide a method for the implementation of the missing alpha and dual exponential PSCs.

The aforementioned standard PSCs are defined as follows [SGGW11].

- The *exponential PSC* is given by:

$$j(t) = \frac{q}{\tau} \mathrm{e}^{-\frac{t-t_m^{(f)}}{\tau}} \Theta(t - t_m^{(f)}), \tag{5.2}$$

  where q is the total electric charge transferred through the synapse, τ is the characteristic decaying time of the exponential function, t_i is the initial time of response to the i-th spike and Θ represents the Heaviside step function.

- The *dual exponential PSC* is introduced when –besides a decaying characteristic time – the specification of finite rising time is desirable. In such a case, two characteristic time constants τ_r and τ_f are used, which control the rising and falling rates, respectively. Then, the dual exponential PSC acquires the form:

$$j(t) = \frac{q}{\tau_f - \tau_r} \left(\mathrm{e}^{-\frac{t-t_m^{(f)}}{\tau_f}} - \mathrm{e}^{-\frac{t-t_m^{(f)}}{\tau_r}} \right) \Theta(t - t_m^{(f)}). \tag{5.3}$$

- An *alpha PSC* is obtained in the limiting case of $\tau_r \to \tau_f$, which transforms

equation 5.3 into an alpha function i.e. a function of the form $f(x) = xe^{-x}$, then:

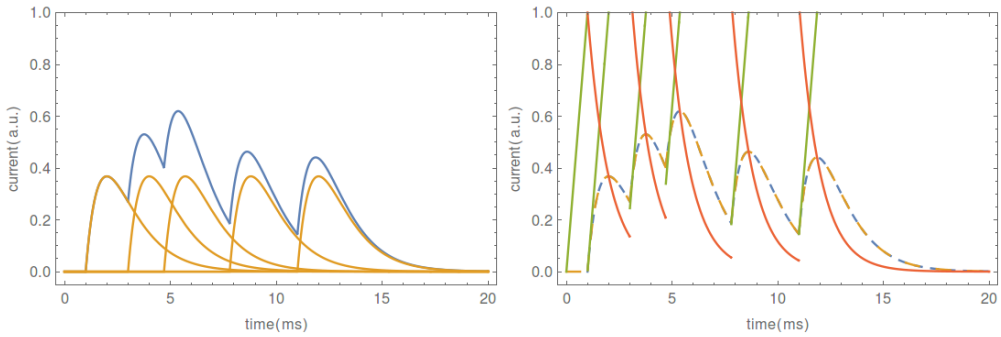
$$j(t) = q\frac{t - t_m^{(f)}}{\tau^2}\left(e^{-\frac{t-t_m^{(f)}}{\tau}}\Theta(t - t_m^{(f)})\right). \tag{5.4}$$

The dependence of equations 5.2, 5.3 and 5.4 on their time constants is shown in figure 5.1.

When considering neuromorphic hardware with limited local memory resources, the feasibility of implementing any PSC depends entirely on its linearizability. The goal is to find an expression that preserves the mathematical form of an individual PSC while accounting for the summations of several incoming spikes. I demonstrate on the following sections how with an adequate use of the Heaviside function piece-wise version of equations 5.3 and 5.4 are able to satisfy the desirable linear condition upon summation. Linearizability implies that one should assign a block of memory (buffer) to each neuron, but not to each synaptic event. Otherwise, the system would be unsustainable. The approach presented here has minimal implications on memory consumption when compared with that of the already implemented exponential function. Importantly, the same strategy can be used to implement alpha or dual exponential elegibility traces for both, plasticity [KTK$^+$16a] and neuromodulation [MPGKF18]. These are essential features for supervised, unsupervised and reinforcement learning in SNNs. The current versions of plasticity and neuromodulation in SpiNNaker, presented in [KTK$^+$16a] and [MPGKF18] respectively, are based on equation 5.2. However, implementing a skewed bell-shaped trace, opens the possibility for ranking the importance of events in a given time window into the past. The formulation of such traces would be identical to the PSCs presented here, so we focus on these for the rest of this chapter.

5.1 Linearisation of Alpha PSC

Let us consider N arbitrary spikes triggering postsynaptic currents $j_i(t)$ at times t_i in a particular neuron α. In general, the spikes are emitted by several presynaptic neurons, but in a point neuron model they are indistinguishable to neuron l. Thus, without loss of generality, we disregard such distinction in what follows and characterize spikes just



(a) (b)

Figure 5.2: a) Alpha function postsynaptic currents (in yellow) generated by individual spikes arriving at arbitrary times. The blue line is the total induced current. b) Linear piece-wise decomposition of the arbitrary postsynaptic current in a). The piece-wise linear component in green and piece-wise exponential component in red are obtained from equations 5.10 and 5.11. The total overlapping of the total current from equation 5.1 with 5.5 and that from the multiplication of 5.10 with 5.11 is shown by the blue and orange dashed lines in b).

by their times, consequently, $\sum_m \sum_f \rightarrow \sum_i$ in equation 5.1 and equation 5.4 becomes:

$$j_i(t) = q\frac{t-t_i}{\tau^2}\left(e^{-\frac{t-t_i}{\tau}}\right)\Theta(t-t_i). \tag{5.5}$$

Figure 5.2a shows the effect of five arbitrary of such PSCs, on the total postsynaptic current $J(t)$ induced by their arrival to the postsynaptic neuron. In the following let us denote $\Theta(t-t_i)$ by Θ_i. Inserting equation 5.5 in 5.1 and assuming $w_{\alpha\beta} = 1$ (see appendix B), we demonstrate that for an arbitrary number of spikes:

$$J(t > t_0) = q\frac{\left(t - \frac{\sum_i t_i e^{\frac{t_i}{\tau}}\Theta_{t_i}}{\sum_i e^{\frac{t_i}{\tau}}\Theta_{t_i}}\right)}{\tau^2} e^{-\frac{t-\tau ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_{t_i})}{\tau}}, \tag{5.6}$$

where we have denoted by t_0 the time corresponding to the first spike. Let us write $\alpha = \sum_i e^{\frac{t_i}{\tau}}\Theta_{t_i}$ and $\beta = \sum_i t_i e^{\frac{t_i}{\tau}}\Theta_{t_i}$, so we can define the auxiliary times $t_\gamma = \frac{\beta}{\alpha}$ and $t_\kappa = \tau ln(\alpha)$. Thus we have:

$$J(t > t_0) = q\frac{(t-t_\gamma)}{\tau^2} e^{-\frac{t-t_\kappa}{\tau}}, \tag{5.7}$$

which has the same form of equation 5.4 but this time with the constants defined

piecewise by the Heaviside functions Θ_i. Notice from equation 5.6 that $t_\gamma = \beta/\alpha$ is undetermined before any spike arrives, resulting in a $0/0$ indetermination and motivating our restriction to $t > t_0$. Clearly, before any spike has been emitted, the PSC due to presynaptic firing is exactly zero. Hence for any arbitrary time we have:

$$J(t) = \begin{cases} q\frac{(t-t_\gamma)}{\tau^2}e^{-\frac{t-t_K}{\tau}} & \text{if } t > t_0 \\ 0 & \text{if } t < t_0. \end{cases} \tag{5.8}$$

Redefining α as $\alpha = \sum_i e^{\frac{t_i}{\tau}}\Theta_{t_i} + (\Theta_0 - \Theta_{t_0})$ allows us to account for the two cases with a single expression, in which case t_γ is well defined for every t and equation 5.8 becomes simply:

$$J(t) = q\frac{(t-t_\gamma)}{\tau^2}e^{-\frac{t-t_K}{\tau}}. \tag{5.9}$$

It is evident now, that in order to implement the alpha PSC in SpiNNaker two buffers need to be updated, one for the linear component:

$$\lambda(t) = q\frac{(t-t_\gamma)}{\tau^2}, \tag{5.10}$$

and another for the exponential component:

$$\varepsilon(t) = e^{-\frac{t-t_K}{\tau}}. \tag{5.11}$$

Equation 5.9 and their components 5.10 and 5.11 have been plotted in figure 5.2b, together with the total current obtained from 5.1 with 5.5. The total overlap of the curves from equation 5.1 and from its piece-wise linear version 5.9 highlights their equivalence. The most important feature to note from figure 5.2b is that at each point the induced current can be obtained by multiplying the values from a linear function and an exponential function, resulting in a generalized alpha function. In other words, the sum of several alpha functions results in a piece-wise alpha function.

Now that we have a linearized form for the total current caused by several alpha postsynaptic responses, let us see how can we implement it in our system. Recall from section 1.2 that SpiNNaker only supports 64K data memory (DTCM) and 32K instruction memory (ITCM)available for each processor, these store the state variables

describing each of the 1024 neurons per core and their updating rules respectively. Thus at any particular time t_n SpiNNaker does not contain explicit information of the past events (e.g., input spike times), it only knows its current state, the general rules that govern its dynamics and some persistent parameters. Furthermore, as a digital system, SpiNNaker runs in discrete time characterized by some time-step Δt. Then, we need to define recurrent relations that allow us to infer the next value of a function based only on its current state and a set of general updating rules, which should also account for the arrival of new spikes. Otherwise, input spike times would have to be saved in memory, quickly consuming any available memory as spikes are generated by the network and limiting the simulation to a very small runtime, while SpiNNaker should be able to run unlimitedly. We use data buffers to save the function value corresponding to the current time t_n, and establish the updating rules as follows. At t_n the buffers corresponding to equations 5.10 and 5.11 are designated by λ_n and ε_n, so we need to determine the values λ_{n+1} and ε_{n+1} for the next time-step. The updating is twofold, an updating rule for the time between spikes where the function is continuously differentiable, and one for the discontinuities at the spikes arrival. In appendix B.1.3 we compute, for both cases, $\Delta\lambda = \lambda_{n+1} - \lambda_n$ and $\Delta\varepsilon = \varepsilon_{n+1} - \varepsilon_n$. Obtaining [1]:

$$\lambda_{n+1} = \lambda_n + \frac{q}{\tau^2}\Delta t \tag{5.12}$$

and

$$\varepsilon_{n+1} = \varepsilon_n e^{-\frac{\Delta t}{\tau}} \tag{5.13}$$

as the updating rules for each continuous segment of the linear and exponential buffers. And:

$$\varepsilon_{n+1} = e^{-\frac{\Delta t}{\tau}}\varepsilon_n + 1 \tag{5.14}$$

with

$$\lambda_{n+1} = \left[\frac{q}{\tau^2}\Delta t + \lambda_n\right]\left(1 - \frac{1}{\varepsilon_{n+1}}\right) \tag{5.15}$$

[1] See section 5.5 for an alternative formulation in the framework of [RD99].

for the updating rules at the discontinuities of the linear and exponential functions. Notice that equations 5.12-5.15 depend only on the current value stored on each buffer and the global parameters q,τ and Δt defined a priori. It is interesting to highlight that λ_{n+1} depends on the updated value ε_{n+1} which is readily available given the fact that at t_n we know ε_n. As a proof of concept, equations 5.12-5.15 have been plotted in figure 5.3 together with equations 5.9-5.11 demonstrating the match between the discrete buffered and the piecewise continuous systems.

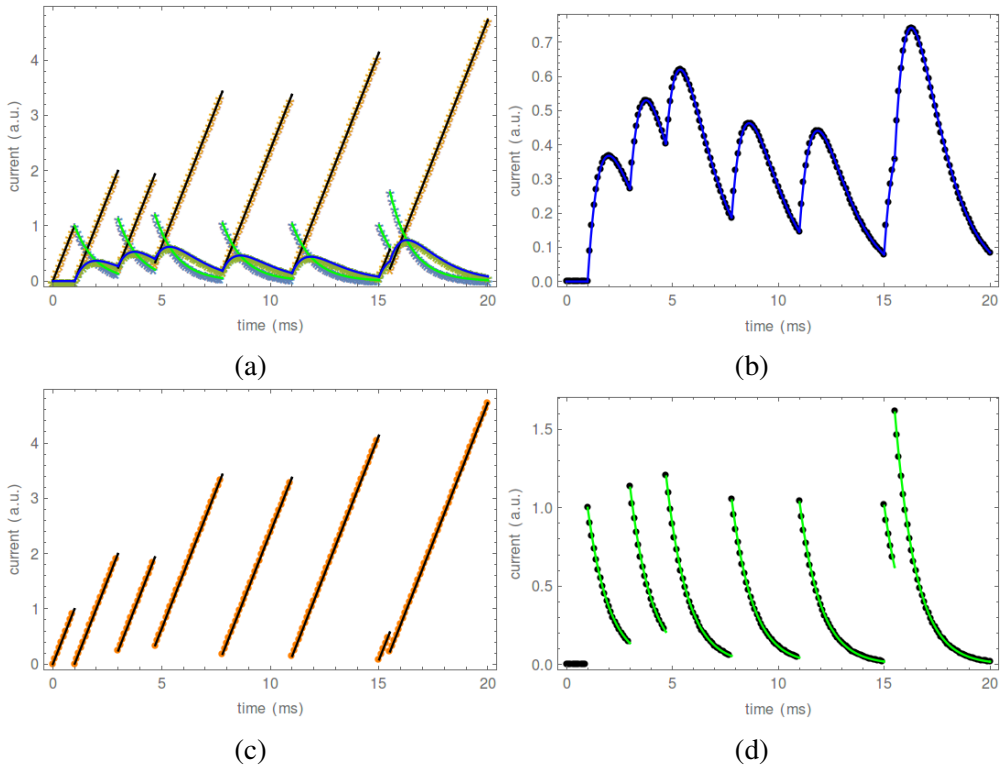


Figure 5.3: a) discrete-time generalized alpha function using linearly and exponentially updated buffers (discrete markers), the piecewise continuous functions were also plotted for comparison (continuous lines). b) shows the matching between the analytical version of the alpha PSC and the buffer updating deduced here. c) linear kernel and d) exponential kernel (also shown in figure a).

5.2 Linearisation Dual Exponential PSC

In the same way, as with the alpha function, we consider N arbitrary spikes arriving at times t_i to our postsynaptic neuron l. Using equations 5.1 and 5.3, the the total current on l at an arbitrary time t can be expressed as (see deduction on Appendix B.2):

$$J(t) = \sum_i j_i(t)$$
$$= \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-\tau_f ln(\sum_i \mathrm{e}^{\frac{t_i}{\tau_f}}\Theta_{t_i})}{\tau_f}} - \mathrm{e}^{-\frac{t-\tau_r ln(\sum_i \mathrm{e}^{\frac{t_i}{\tau_r}}\Theta_{t_i})}{\tau_r}}\right). \tag{5.16}$$

Let us define $\alpha = \sum_i \mathrm{e}^{\frac{t_i}{\tau_f}}\Theta_{t_i}$ and $\beta = \sum_i \mathrm{e}^{\frac{t_i}{\tau_r}}\Theta_{t_i}$ So we can write $t_\gamma = \tau_f ln(\alpha)$ and $t_\kappa = \tau_r ln(\beta)$ as the generalized times for the dual exponential PSC. Now equation 5.16 takes the form:

$$J(t) = \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-t_\gamma}{\tau_f}} - \mathrm{e}^{-\frac{t-t_\kappa}{\tau_r}}.\right) \tag{5.17}$$

Equation 5.16 defines N+1 regions of time characterized by the respective values of Θ_i in each zone. This implies that for an arbitrary region $t_{j-1} < t_j < t_{j+1}$, $\alpha = \sum_i^j \mathrm{e}^{\frac{t_i}{\tau_f}}$ and $\beta = \sum_i^j \mathrm{e}^{\frac{t_i}{\tau_r}}$.

We need two buffers again to implement this PSC in the SpiNNaker toolchain. These are given by:

$$\varepsilon^\gamma(t) = \mathrm{e}^{-\frac{t-t_\gamma}{\tau_f}}$$
$$\varepsilon^\kappa(t) = \mathrm{e}^{-\frac{t-t_\kappa}{\tau_r}} \tag{5.18}$$

These have the same form of the exponential buffer in the alpha PSC of the last section. Thus, the updating rule for each continuous segment of the generalized dual exponential function is:

$$\varepsilon^\gamma_{n+1} = \varepsilon_n e^{-\frac{\Delta t}{\tau_\gamma}}$$
$$\varepsilon^\kappa_{n+1} = \varepsilon_n e^{-\frac{\Delta t}{\tau_\kappa}}. \tag{5.19}$$

And the updating rule at the discontinuities is given by:

$$\varepsilon^{\gamma}_{n+1} = e^{-\frac{\Delta t}{\tau_\gamma}}\varepsilon_n + 1$$
$$\varepsilon^{\kappa}_{n+1} = e^{-\frac{\Delta t}{\tau_\kappa}}\varepsilon_n + 1 \tag{5.20}$$

for . Once again we evidence how equations 5.19 and 5.20 depend only on the current value stored on each buffer and the known global parameters q, τ and Δt. Figure 5.4 shows the behavior of the buffers, equation 5.18, updated with equations 5.19 and 5.20, as well as how these match the piecewise continuous version of equation 5.16.

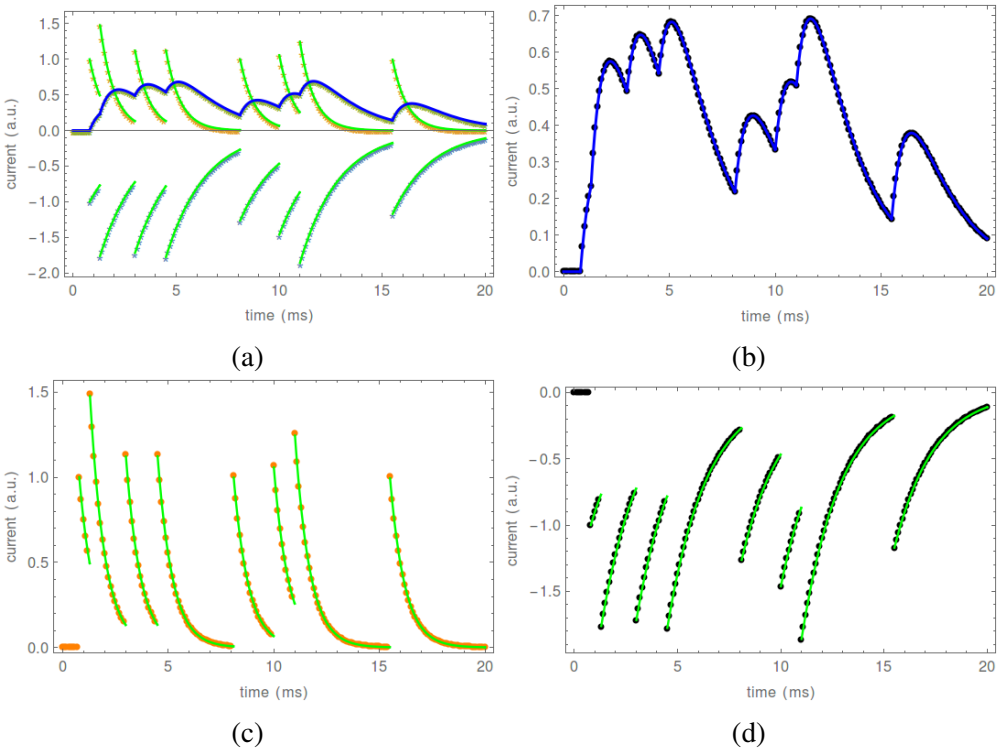


Figure 5.4: a) discrete-time generalized dual exponential function using two exponential updateding buffers (discrete markers), the piecewise continuous functions were also plotted for comparison. b) shows the matching between the analytical version of the dual exponential PSC and the buffered version described here. c) and d) show the two exponential kernels with their respective buffers.

5.3 Linearisation of Rectangular PSC

A rectangular PSC can be expressed as:

$$j(t) = j_0(\Theta_i - \Theta_{i+\tau}) \tag{5.21}$$

where τ is the span of the response and j_0 its intensity, which remains constant along τ. With N spikes arriving at times t_i the total response current is given by:

$$J(t) = \sum_i j_0(\Theta_i - \Theta_{i+\tau}) \tag{5.22}$$

The implementation of such PSC in SpiNNaker requires a temporally local measurement of time to control the pulse duration τ. As there is no global clocking in the system, an implementation of this rather abstract PSC can be achieved by a using the ceiling function $\lceil\rceil$ together with a linear function. We need to rewrite 5.22 as:

$$J(t) = \sum_i j_0\lceil(t_i - t)\tau^{-1} + 1\rceil(\Theta_i - \Theta_{i+\tau}) \tag{5.23}$$

where the function $(t_i - t)\tau^{-1} + 1$ is a decreasing linear function which equals 1 at t_i and goes to 0 at $t_i + \tau$, this function leaves 5.22 intact, but allows SpiNNaker to measure the finite duration of the square pulse. We will then need a buffer for the linear function:

$$\lambda(t) = (t_i - t)\tau^{-1} + 1 \tag{5.24}$$

which is set to the value 1 at the arrival of the first spike and is then updated –until a new spike arrives– according to:

$$\lambda_n = \lambda_{n-1} - \frac{\Delta t}{\tau} \tag{5.25}$$

where the subindex n run over discrete steps with separation Δt. At the arrival of a new spike the buffer is updated as:

$$\lambda_n = \lambda_{n-1} + 1 \tag{5.26}$$

5.4 Implementation

Following our model Dr. Oliver Rhodes, one of the core SpiNNaker software developers, implemented the alpha and dual exponential PSCs. These are now released as part of the SpiNNaker tool-chain [Tea20a]. The code for the alpha PSC is publicaly available at `https://github.com/SpiNNakerManchester/sPyNNaker/blob/master/neural_modelling/src/neuron/synapse_types/synapse_types_alpha_impl.h` [Tea20b] and that for the dual PSC can be found in `https://github.com/SpiNNakerManchester/sPyNNaker/blob/master/neural_modelling/src/neuron/synapse_types/synapse_types_dual_excitatory_exponential_impl.h` [Tea20c].

5.5 Previous Work

After the development exposed in the previous sections, we found ours to be a special case of the more general formulation of S. Rotter and M. Diesmann [RD99]. They found that time-invariant linear systems can be iteratively integrated exactly, while these evolve over a discrete-time regular grid. Such systems are described by a first-order n-dimensional linear differential equation of the form:

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{x}. \tag{5.27}$$

The n-dimensional vectors $\mathbf{y}$ and $\mathbf{x}$ correspond respectively to the evolution of the system and its input. In turn, the input-output relation is encoded in the square matrix $\mathbf{A}$. The solution to equation 5.27 is given by:

$$\mathbf{y}(t) = e^{\mathbf{A}(t-s)}\mathbf{y}(s) + \int_{s+}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{x}(\tau)d\tau \tag{5.28}$$

Where the initial value $\mathbf{y}(s)$ and input response govern the time evolution of $\mathbf{y}$. Notice that all previous dynamics is encoded on the initial value $\mathbf{y}(s)$. Thus, for generalised inputs $x(t) = \sum_k \delta(t - t_k)$, the simulation of $\mathbf{y}$ over regular discrete times $t_k = k\Delta t$ can be done recursively in an exact manner through:

$$\mathbf{y}_{k+1} = e^{\mathbf{A}\Delta t}\mathbf{y}_k + \mathbf{x}_{k+1} \tag{5.29}$$

Where $e^{\mathbf{A}\Delta t}$ is a matrix exponential. The fact that the matrix exponential is constant for constant Δt and that $\mathbf{y}_{k+1}$ can be written in terms of $\mathbf{y}_k$ and any input at timestep t_k,

implies the linearisability algebraically demonstrated in the previous sections. Rotter and Diesmann demonstrate the evolution of $\mathbf{y}_k$ for exponential damped, oscillatory and polynomial systems by considering the fundamental variations for $\mathbf{A}$. In particular, the alpha and beta functions are obtained as solutions to:

$$\ddot{\boldsymbol{\eta}} + (a+b)\dot{\boldsymbol{\eta}} + (ab)\boldsymbol{\eta} = 0 \tag{5.30}$$

Subject to $\boldsymbol{\eta}(0) = 0$ and $\dot{\boldsymbol{\eta}}(0) = \boldsymbol{\eta}_0$. Alpha and beta functions are obtained for $a = b$ and $a \neq b$ respectively. Note that 5.30 is a second order differential equation, however, high-order differential equations can be transformed into first-order differential equations by adequate substitution of variables. In terms of equation 5.27, it is thus possible to write:

$$\boldsymbol{y} = \begin{bmatrix} b\boldsymbol{\eta} + \dot{\boldsymbol{\eta}} \\ \boldsymbol{\eta} \end{bmatrix} \tag{5.31}$$

$$\boldsymbol{x} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5.32}$$

$$\boldsymbol{y}(0) = \begin{bmatrix} \boldsymbol{\eta}_0 \\ 0 \end{bmatrix} \tag{5.33}$$

$$A = \begin{bmatrix} -a & 0 \\ 1 & -b \end{bmatrix} \tag{5.34}$$

The corresponding matrix exponential for the alpha and beta functions are:

$$\begin{bmatrix} e^{-a\Delta t} & 0 \\ \Delta e^{-a\Delta t} & e^{-a\Delta t} \end{bmatrix} \tag{5.35}$$

and

$$\begin{bmatrix} e^{a\Delta t} & 0 \\ \frac{1}{b-a}\left(e^{a\Delta t} - e^{b\Delta t}\right) & e^{-b\Delta t} \end{bmatrix} \tag{5.36}$$

The fact that this method further allows the implementation of sinusoidal and damped oscillations opens further possibilities for neural coding in neuromorphic hardware, see for example [FS19, BW12].

5.6 Conclusions

This chapter demonstrates how the mathematical shape of the standard alpha and dual exponential postsynaptic currents, which govern the integration of spiking input to a neuron, remains when several inputs are added, which is, they are linearizable. A model suitable for SpiNNaker's discrete-time dynamics and limited memory availability was presented with minimal implications for performance. This is because the fraction of instructions memory assigned for synaptic updating is a modest fraction of the total memory assigned to a neural model [RBB$^+$18], the cost of the new currents consists on the use of an extra memory buffer per neuron, as well as the instructions for its updating. Both postsynaptic currents have been implemented as a novel feature of the machine and are publicly available for use through the sPyNNaker software.

The formulation is useful also to implement a new kind of plasticity and neuromodulation traces. Using alpha or dual-exponential traces will allow the control of the point in time, which makes the maximum contribution to synaptic changes or intrinsic excitability. The relative value of the time constants defining the curves controls the summit of the traces. The implementation and analysis of the benefits for learning in SNNs are left as future work.

Chapter 6

Intrinsic Currents Generated by Voltage-gated Ion Channels

Chapters 2 and 4 have demonstrated the advantages of spiking point neuron models for the development of efficient stochastic computations in neuromorphic hardware for the solution of mathematical problems. However, when modelling biological networks, some intrinsic features of synaptic integration, excitability and plasticity are not accountable for unless the modulation of the intrinsic excitability of the neuron caused by the interplay between transmembrane ionic currents is included in the formulation. Such currents originate from the collective kinetics of ions across ensembles of transmembrane ligand- and voltage-gated ion-channels and are known to play a significant role both in cognitive tasks and in medical conditions. However, modelling intrinsic currents is computationally expensive because their formulation involves nonlinear coupled differential equations, these are described by voltage-dependent conductances and the voltage-dependence appears in exponential and divide functions. In chapter 5, the exact integration of exponential functions was possible due to the regular time grid over which the simulation happens, the same strategy is used in neural simulators and neuromorphic hardware for the integration of exponential postsynaptic currents and membrane voltage [RBB$^+$18]. Here, however, the membrane voltage can assume arbitrary and nonmonotonic values with time, making it not possible to harness the bounding of computational resources achieved through exact integration in a discrete-time regular grid. Here, we present preliminary results on the implementation of neuronal voltage-dependent intrinsic currents in SpiNNaker. Such capabilities in a million-core digital neuromorphic system, open new possibilities to bridge modelling done in a diversity of scales, helping to understand the brain functioning from first principles. In SpiNNaker, the largest and

most versatile neuromorphic machine to date, the hardware architecture still imposes hard constraints on a model implementation, compromising its feasibility. In particular, one counts only with floating-point arithmetic and limited local memory for instructions (32 KB). Hence, the compactness, stability and accuracy of the software should be carefully analysed. The development presented here uses a fixed-point implementation of the exponential function available in the SpiNNaker software stack (*expk*). We also approximated the voltage-dependence of the conductance with piece-wise linear and polynomial functions, however, such strategy results more expensive than *expk*, while preserving the same error range. Hence, such an approach was abandoned.

6.1 Background and Motivation

Given the unique complexity and dimensionality of the brain, it is essential to choose an adequate level of abstraction which disregards a large number of variables while preserving the relevant details affecting the phenomena under study. Point spiking neuron models are one such abstraction which captures the dynamics of the action potentials (spikes) generation, transmission and integration, disregarding intrinsic properties which emerge from a complex molecular dynamics happening both, inside the cell and across the plasma membrane [GKNP14]. The description of such intrinsic properties, when possible, requires several coupled nonlinear differential equations per neuron [H$^+$01]. Hence, disregarding them allows the simulation of extensive networks of spiking neurons and their dynamics (the current world record being 1.86×10^9 neurons connected by a total of 11.1×10^{12} synapses) [KSE$^+$14, vARS$^+$18, Fur16a]. This simplified neuron model approach has evolved our understanding of diseases [ACM12, MHB08, MB09] cognitive processes [WBK10a, WBK10b, Flo05] and task performing behaviour [ESC$^+$12, FGF17]. Nonetheless, some aspects of cognition, neurological disorders, plasticity and neuromodulation, depend critically on the neglected phenomenology [RG02, LLW$^+$04, FJ05b, FMJ04, DRT99, CT04, BW80, BLR$^+$02, ARTD00, ALS82, TAM94, vWvHW04, ZL03]. Thus, although some fine-grained details, like those related to energy consumption, energy maintenance or nutrients recycling, may not be very prominent for information processing and diseases studied at the network level, others are, and at some point should be taken into account. Advancement has been done with dynamically richer models like that of [Izh03]. These, however, are indirectly related to the ion's dynamics and are not adaptive, changing firing behaviour according to a pre-defined set of parameters. Despite the undeniable usefulness of these

latter models, it remains desirable to implement models that can harness cellular and molecular data from single neuron recordings and enable the exploration of their effect at network levels which are implausible experimentally. In the following, we keep some well known biological and theoretical details explicit aiming to a self-contained chapter which gives enough information to both the neuroscientist and the computer engineer. In this way, the underlying assumptions, and hence the validity of the model, are clear for the former while the latter can reason about the limitations, implementation constraints and future extensions to the software and hardware platforms.

On a first approximation, both the action potentials and the intrinsic excitability of a neuron are originated from the interplay between diverse transmembrane ionic currents [SGGW11]. The underlying ion kinetics depends on factors such as binding of ligand molecules to membrane receptors, membrane voltage, temperature, mechanical deformation and ion concentrations [H^+01]. The ion transport across the membrane is mediated by iontophoretic proteins (ion carriers) and can be passive – along the electric or concentration gradients– and active – spending energy (e.g. ATP) against those gradients. The main passive transporters are the voltage-gated and the ligand-gated ion-channels, these are pore-forming transmembrane proteins that control what ions enter and leave the neuron at a particular membrane voltage or after the binding of some intracellular or extracellular ligand molecule (or second messengers) respectively [PK87]. This passive transport can drastically influence the firing behaviour of the cell, contributing to the electrical identity of each neuron type [Lli14, Lli88]. For example, the interplay between currents governing a fast-spiking response and those responsible for slow modulation of activity, generate the high-frequency bursting characterising the low-threshold bursters, hippocampal pyramidal neurons located in the area CA1 [SAKY01].

Single neurons typically contain tens of thousands of ion channels spanning more than 12 types per neuron on average [Sig14, H^+01]. Furthermore, genetic studies in humans have identified over 140 members for the superfamily of genes expressing voltage-gated ion-channels giving rise to a total of around 5000 ion-channel types [GSMG07, H^+01]. The diversity of channels causes that even under the same connectivity and neurotransmitter bath, the response of two neurons could be vastly different. Taking into account the demonstration by [IE08] that even a single spike suffices to change the global state of a network of spiking neurons, it becomes evident that a change in the neuronal response implies a potential change in the network trajectory through its state space. Such variability is disregarded in the simplified spiking neuron

models, somehow conflicting with the law of no interchangeability of neurons (Llinás's law) [Lli88]. In this chapter, we explore the possibility of including neuron variability through explicitly augmenting a spiking neuron model with voltage-dependent intrinsic currents. As for the role of active transporters, like the sodium-potassium pump and the sodium-calcium exchanger, it is well covered abstractly by standard spiking neuron models. For example by the existence of a resting membrane potential or its recovery after the refractory period [SGGW11, GKNP14]. Hence the detailed implementation of active transporters is of less importance and potentially unnecessary. Thence, the implementation here offers a trade-off between biological detail and computational feasibility, advancing towards bridging the gap between neuromorphic computing and neurophysiology. It enables the use of the experimental data from voltage-clamp protocols, standard in the study of real neurons, to fit the activation and inactivation parameters readily available in the next version of SpiNNaker. The experimenter will then able to explore the network behaviour caused by the specificity of the neuronal dynamics of recorded neurons. With further development, the intrinsic currents and voltage-clamp simulations presented here for SpiNNaker will be made part of SpyNNaker [RBB$^+$18], the standard software backend for PyNN simulations on SpiNNaker that we used in chapter 2. The integration will make it usable along with the diverse functionalities already available on the machine.

Historically, our understanding of intrinsic currents began when K. S. Cole and R. F. Baker [CB41a, CB41b] characterised the squid giant axon electrical response to alternating currents, identifying its inductive reactance and the rectifying character in terms of ion permeability, leading in the subsequent years to the confirmation of the ionic theory of membrane excitation. Later, Hodgkin and Huxley [HH52] developed the voltage clamp protocol to analyse the potassium and sodium ion currents on the squid giant axon and explained their origin through a model of voltage-gated membrane pores built from independent particles. Their mathematical description explained the axon's generation of action potentials from a precise interplay between voltage-dependent sodium, potassium and leak currents. Further, electrical noise measurements [KM70], together with the development of the patch-clamp technique [SN84], which allows performing voltage clamps in tiny patches of cell membranes, as small a to include a single channel, confirmed the existence of the ion channels and their relation to the previously observed ionic currents [see for example VB91]. Since then, voltage-clamp, single-ion channel and whole-neuron recordings have allowed the identification of numerous ion-channel types and ionic currents which affect the excitability of neurons

well beyond the action potential generation mechanism.

The kinetics between the opening and closing states of a single voltage-gated ion channel obeys a stochastic Poisson process with the open probability influenced by the membrane potential. The channel activity averaged over the thousands of channels existing along the neuronal membrane or equivalently over several measurements on the same channel, reveals a qualitative behaviour resembling that of the corresponding intrinsic current. Hence, intrinsic currents can be thought of as an ensemble manifestation of the ion-channels dynamics. The electrophysiological studies during the '80s and '90s improved after Hodgkin and Huxley [HH52], allowed the identification of diverse families of intrinsic currents involved in for example the rhythmic oscillations in relay cells [HM92, DBS93, DNUH98]; the electrical resonance at theta frequencies in rat hippocampal pyramidal cells [HVS02]; and the shaping of intrinsic firing of rat abducens motoneurons [RCA$^+$03]. The data from these experiments constitutes an integral part of our current understanding of brain functioning. However, the field of neuromorphic computing has focused more strongly in neuroanatomical and functional maps of the neural networks in the brain, a development that has reached a sufficient level of maturity to make it essential to explore further complexities and the limits of these machines. Driving new developments, as well as the next generation of the physical realisations of the existing paradigms. Here, we aim to leverage neuromorphic computing one-level down to the modelling of cellular variability while seizing the massive parallelism offered by SpiNNaker. It is worth mentioning that neural variability has been accounted for before in SpiNNaker through the use of the Izhikevich neuron model, it offers efficiency while keeping some of the rich dynamics of the original Hodgkin-Huxley model. However, the neuron-type of an Izhikevich neuron is controlled through 4 parameters which are indirectly related to the cellular phenomenology and remain constant through the simulation runs. Though one could hard-code runtime changes in the neuron-type, these will not depend on the network activity. On the other hand, real neurons adapt to the network activity through intrinsic currents, for example in homeostasis and intrinsic plasticity, switching between a repertoire of dynamics characterised by both different firing patterns and different sub-threshold dynamics.

In the following section, we review the previous work on implementing intrinsic currents on neuromorphic hardware, then, in section 6.3, we use the known kinetic scheme descriptions of ion-channel dynamics as a Markov process to derive the voltage and time dependence of a generic intrinsic current in a form suitable for implementation

in SpiNNaker. Our contribution here is on setting the stage for a software implementation on the SpiNNaker hardware by assembling several converging theoretical frameworks. Ideally, the theoretical background sets a principled formulation with broad application and few assumptions. The limitations would then come from the hardware specification, unraveling its limitations and directions fro future hardware development. The model leverages Hodgkin-Huxley models but uses rate coefficients as described by the transition state and Kramer's theories of reaction kinetics. The goal is to avoid the need for implementing the diversity of freely defined functions for fitting such coefficients while allowing reuse and fitting of experimental data. Some collective effects from ligand-gated ion-channels are already included implicitly on the plasticity, neuromodulation, and synaptic transmission implementations of the SpiNNaker Software [KTK$^+$16b, MPGKF18, RBB$^+$18]. Hence, a more detailed approach to ligand-gated ion-channels is also left for subsequent work.

6.2 Previous Work

The first neuromorphic hardware implementation of intrinsic currents is perhaps the one by M. Mahowald and R. Douglas [MD91]. They recognized the similarities between the sigmoid shape of the current-voltage relation of a differential pair formed by complementary metal-oxide-semiconductor (CMOS) transistors and the conductance-voltage relation of the steady-state activation and inactivation variables describing ion-channel dynamics in cell membranes. Using a very-large-scale integration (VLSI) process, they fabricated a two-compartment CMOS circuit that was able to produce some of the observed electrophysiological responses of real neurons under current-clamp conditions, as well as the modulation of the firing rate by both the slow potassium current I_{KA} and the calcium-dependent potassium after-hyperpolarizing current I_{AHP}. A conductance transistor represented each conductance, differential pairs controlled the kinetics, the maximum activation/inactivation was set by bias transistors, and the knee transistor of each pair determined the activation voltages. The neuronal membrane was implemented with a fixed capacitor and a variable leak conductance to account for voltage-independent conductances. The hardware model accurately represented the voltage dependence of the steady-state with a power dissipation from the whole circuit of about 60 μW. However, the time dependence implemented by using follower-integrators with variable time constants to act as a low-pass filters for the membrane voltage, do not correspond to the the bell-shaped voltage-dependent time constant characterizing

intrinsic currents.

Later, D. Dupeyron et al., [DMD+96] also implemented an analogue circuit to compute the current generated by Hodgkin Huxley conductances. They used a full-custom ASIC (2 μm Bipolar-CMOS) to approximate the exponentials-based sigmoids defined by the steady-state activation and inactivation variables raised to their respective powers, by linear sigmoids of these variables where the offset and slope were modified to make it approximately equivalent to the original functions through $V_{offset} = V_{offset_0} - V_{slope_0} \ln(\sqrt[n]{2} - 1)$ and $V_{slope} = V_{slope_0} \cdot \sqrt[n]{2}/2n(\sqrt[n]{2} - 1)$. This strategy allows the reduction from $i_{ion} = g_{max} \cdot m^p \cdot h^q \cdot (V_{mem} - V_{equi})$ to $i_{ion} = g_{max} \cdot m \cdot h \cdot (V_{mem} - V_{equi})$. Their chip implemented three circuit blocks able to represent any pair of activation or inactivation variables (e.g., Na+, K+, leakage). This was done by harnessing the sigmoidal shape of the transmission characteristic of a bipolar differential pair and the ability of a MOS transistor to implement offsets. The kinetics was implemented through a subtractor and an integrator operational amplifiers, achieving a maximum τ_m of $5\,ms$ and maximum g_{max} of $3 \times 10^6\,S$. Again, such a model reproduces some of the qualitative behaviours of Hodgkin-Huxley currents and is energy efficient. Unfortunately, the hardware imposes limits on parameter values. The chip cannot model the voltage dependence of τ_m, which plays a fundamental role in intrinsic currents not involved in the action-potential generation, and it is sensible to noise.

More recently, [MHSM12] presented an implementation of a four-channel bursting silicon neuron (I_L, I_T,I_K and I_{leak}, note that I_T and I_L here only included m and were raised to the power of 1). By using a few transistors, a current-reuse technique and a subthreshold region operation of MOSFETs they achieved ultra-low power consumption. Similarly to [MD91], intrinsic currents were implemented by a sigmoid-function circuit based on a MOS differential pair, a subthreshold MOSFET-based differential amplifier for both I_L and I_K, and a gate controlled MOSFET differential amplifier for the fast-response sigmoid function in I_T. A linear transconductance was used to control the output current with changes in the membrane voltage. Setting the currents time constants, through the capacitor of a log-domain filter, allowed the generation of four different neuronal behaviours: spiking, spiking with latency, bursting, and chaotic signals. The circuit was built using a 0.13 μm standard process, it comprises 43 transistors, and consumes 43 nW.

To the author's knowledge, the only complete neuromorphic implementation of the thermodynamic model of voltage-dependent ion-channel currents, including voltage-dependence in both the steady-state activation, inactivation and in the respective time

constants, is that of K. M. Hynna and K. Boahen [HB07]. Previous hardware proposals had avoided the complexity introduced by the voltage dependence on τ_m to save area of the silicon die. Although such approach allows more neurons per chip, the dependence on V can not always be disregarded, as pointed out in [HB07], V causes an order of magnitude change on τ_m for the case of $I_{Ca(T)}$ in thalamic relay cells and underlies its bursting behaviour.

K. M. Hynna and K. Boahen [HB07] used 8 transistors and harnessed the isomorphism between energy barriers in a transistor and those of the neural membrane in section 6.3. Because the Boltzman distribution describes both, their implementation was developed in terms of reaction rates.

Other neuromorphic designs for intrinsic ion-channel and synaptic variability are based on carbon nanotubes transistors [MP11]. However, this was not realized physically and did not model the intrinsic currents explicitly. Instead, included both possible sources of neural variability, chaos and noise, by embedding either Gaussian noise or a chaotic signal generator into the synaptic and axon hillock circuits, preserving no relation to electrophysiological data from intrinsic currents.

All the works above present high energy-efficiency and a small circuit area. This works well for single neurons or very small networks. However, none of these report on the use or testing of scaled versions of the models. This may be because of sensibility to noise and miss-match, which may seriously limit the scalability of analogue circuits with detailed models of neuronal dynamics. Analogue neuromorphic hardware for large or very large networks are restricted to simplified spiking neuronal models precisely because of that. The SpiNNaker architecture paradigm offers an advantage here, the flexibility of a digital implementation of the neuron models together with the neuromorphic asynchronous massive parallelism, implies that large networks can be built despite the complexity of the model as long as the model fits the instructions memory i.e., the event-driven computation makes the system globally agnostic to the local dynamics allowing separation of scales.

6.3 Model of Intrinsic Currents

In order to derive a principled theoretical model of the electrophysiological currents described in section 6.1, we take a bottom-up approach beginning from the dynamics of a single channel. There are two advantages to this approach. Firstly, experimental data obtained at the molecular and cellular levels may be used together with our

implementation. Secondly, we set the basis for more detailed implementations in future hardware, having fewer constraints and more resources.

6.3.1 Time-Dependence of Channel Dynamics

A voltage-gated ion channel is a complex transmembrane protein composed of a pore-forming substructure, a group of voltage-sensing domains (VSDs) which control the pore aperture, and in some cases one or more inactivating amino acid-protein conjugates which are also voltage-sensitive and intracellularly block the ion-channel pore. Both are made up of a few protein segments composed of thousands of atoms. Given the size and structural complexity of ion channel proteins, any atomistic first-principles approach becomes intractable due to the huge size of the associated Hilbert space, as well as the high temperature of biological systems respect to the ground state. Even a coarse-grained molecular dynamics becomes practically intractable (although free energy profiles of the channel can be obtained in this way as input for kinetic formulations) [Sig14]. Hence we resource to a phenomenological approach [H^+01, DH10]. Each VSD has the ability to undergo reversible voltage-dependent conformational changes $VSD_a \rightleftharpoons VSD_d$ between activated and deactivated positions, so that the channel undergoes transitions $X_i \rightleftharpoons X_j$ between conformational states $X = \{X_i, X_2, ..., X_n\}$. In general, only those channel conformations in which all VSDs are in the activated state and the inactivation segment is not blocking the pore will allow ions to pass through, we call these the *open states* O_i and label the other and the inactivated configurations as *closed states* C_i. Besides interacting with the electric field across the membrane, which acts as a driving force, the channel is also subject to thermal fluctuations which cause it to undergo stochastic transitions. We can then resort to a probabilistic description of the ion channel dynamics. Assuming that the probability $P(X_i \to X_j)$ of transitioning to a new conformational state of the protein depends only on the current state, which has the probability $P(X_i, t)$ of being X_i at time t, one can describe the dynamics of a single channel as the Markov process,

$$X_1 \underset{\alpha_1}{\overset{\beta_1}{\rightleftharpoons}} X_2 \underset{\alpha_2}{\overset{\beta_2}{\rightleftharpoons}} ...X_n \underset{\alpha_n}{\overset{\beta_n}{\rightleftharpoons}} X_{n+1}. \tag{6.1}$$

Here, $\alpha_i = P(X_i \to X_{i+1})$ and $\beta_i = P(X_{i+1} \to X_i)$ are the forward and backward transition probabilities between two adjacent states in the Markov chain. Notice that the Markovian assumption is less limiting than it appears at first, as one can always

generalise the state definition to include several conformations per state so that the assumption is satisfied. We are interested in the temporal evolution of $P(X_i,t)$. Following 6.1, the change in $P(X_i,t)$ will have a positive contribution from the probability that the channel was in the neighbouring states X_{i-1} or X_{i+1} and makes a transition to X_i and a negative contribution from the probability that it was in X_i and transitions to one of the neighbouring states, this is given by the *Master equation* [DH10]:

$$\frac{dP(X_i,t)}{dt} = \sum_{j=1}^{n} P(X_j,t)P(X_j \to X_i) - \sum_{j=1}^{n} P(X_i,t)P(X_i \to X_j). \tag{6.2}$$

Notice that because the process is Markovian, several transition probabilities are expected to be zero, though one still has one equation from 6.2 for each of the $n+1$ states of the process. In general activation and inactivation may be coupled, and the transition coefficients may depend on the membrane potential, as well as on the channel state. In such a general case, equation 6.2 is computationally too expensive for SpiNNaker and requires data from single-channel recordings. Here we consider the simpler case of independence between activation states A, and inactivation states I, as well as voltage-dependent or constant transition coefficients. Furthermore, we consider the VSDs to be independent of each other. This is analogous to [HH52] assumption that the channel opening is governed by a small number of conditionally independent gating particles. In such a case, the activation states can be associated with a sequentially decreasing number of VSDs in the activated configuration, while taking into account the corresponding multiplicity. Considering a single open state and hence n nonconducting conformations, the respective probabilities are given the probability mass function of a binomial distribution multiplied by the deinactivation probability (when considering a single inactivation segment):

$$P(A_i) = \binom{n}{i} p_A^i (1-p_A)^{n-i} p_I \tag{6.3}$$

$$P(I_i) = \binom{n}{i} p_A^i (1-p_A)^{n-i} (1-p_I). \tag{6.4}$$

Here, $\binom{n}{i} = \frac{n!}{i!(n-i)!}$, p_A is the probability of a VSD being in the activated conformation, p_I is the probability of an inactivating particle being in the deinactivating conformation, and n is the total number of VSDs, here equal to the number of closed states A. Accordingly, the transition coefficients can be expressed in terms of the coefficients α and β of an individual VSD transition times the number of VSDs available for the transition in

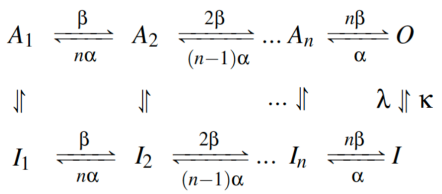


Figure 6.1: Markov state diagram for the ion channel kinetic model.

each state. Hence, following an $n : n-1 : n-2 : ... : 1$ ratio in the forward direction and a $1 : 2 : 3 : ... : n$ ratio in the backward direction. Under these assumptions and a single inactivating segment, one can rewrite equation 6.1 as the restricted Markov model in figure 6.1.

Where λ and κ are the forward and backward transition coefficients for the inactivation segment and apply to all vertical arrows. Top states in figure 6.1 correspond to the unblocking configuration of the inactivating particle and bottom to the blocking or inactivated states, the presence of inactivating transitions to and from all activating states is due to the independence assumption between activation and inactivation. In some cases, as is the case of some Na^+ channels, the inactivating state is reachable only from some of those states (O for example). Such constraint can considerably increase the number of parameters to be fitted and hence the number of coupled rate equations describing the model. The next generation of SpiNNaker (section 1.2), could handle some of this more complex Markov models, but the independence assumption should be held for SpiNNaker 1.

In figure 6.1 the leftmost states are those where the n VSDs are in the closed state. The open state (rightmost top) corresponds to the conducting conformation in which all VSDs are positioned to open the pore, and no inactivation exists. According to equation 6.3, $P(O) = p_A^n p_I$. It is easy to verify that if q inactivating segments exist, $P(O) = p_A^n p_I^q$. Furthermore, following the Master equation 6.2:

$$\frac{1}{p_I p_A^n}\frac{dP(O,t)}{dt} = \frac{n}{p_A}\frac{dp_A}{dt} + \frac{1}{p_I}\frac{dp_I}{dt} \tag{6.5}$$

$$= \frac{n}{p_A}(\alpha(1-p_A) - \beta p_A) + \frac{\lambda}{p_I}(1-p_I) - \kappa. \tag{6.6}$$

Thus:

$$\frac{dp_A}{dt} = \alpha(1-p_A) - \beta p_A \tag{6.7}$$

$$\frac{dp_I}{dt} = \lambda(1-p_I) - \kappa p_I. \tag{6.8}$$

The problem has then been reduced to a single equation for activation and one for inactivation [SGGW11]. Here, we only need a tuple of backward and forward coefficients for each equation. As neuromorphic hardware evolves to support more expensive operations, more general versions of 6.7 and 6.8 derived from the Master equation 6.2 can be considered.

We count now with a probabilistic description of the dynamics of single ion channels. Consider now the cellular membrane of a neuron. The thousands of channels present on it will constitute a statistical ensemble in which, by the law of large numbers, the fraction m of VSDs in the activated states converges to p_A (the activation probability for a single VSD). Likewise, the fraction h of inactivating segments in the unblocking conformation converges to p_I. Equations 6.9 and 6.10 correspond to the microscopic version of 6.7 and 6.8 and describe voltage-dependent ionic currents across the membrane:

$$\frac{dm}{dt} = \alpha(1-m) - \beta m \tag{6.9}$$

$$\frac{dh}{dt} = \lambda(1-h) - \kappa h. \tag{6.10}$$

These are Hodgkin Huxley models, but clearly derived from Markov models. Thus, suitable for extensions, for taking into account the ion-channel inverse problem [CD06], to allow an easy integration of activation-deactivation dependence, as well as to include drug interactions. All of these may be possible in future neuromorphic hardware, like SpiNNaker 2. Hence, though SpiNNaker 1 is better suited for restricted Markov models, as availability of resources increases, unrestricted Markov models and the corresponding terms from the Master equation (6.2) may be included.

In order to obtain expressions suitable for experimental fitting, it is conventional to

consider the steady-state transition rate (when $t \to \infty$) for a given voltage V, defined by:

$$m_\infty(V) = \frac{\alpha(V)}{\alpha(V)+\beta(V)} \tag{6.11}$$

$$= \frac{1}{1+\beta(V)/\alpha(V)}, \tag{6.12}$$

(note it has the shape of a sigmoid curve) and the associated time constant

$$\tau_m(V) = \frac{1}{\alpha(V)+\beta(V)}. \tag{6.13}$$

The same shape holds for h. Then, equations 6.9 and 6.10 can be expressed as:

$$\frac{dx}{dt} = \frac{x_\infty - x}{\tau_x} \mid x \in \{m,h\}. \tag{6.14}$$

Where it becomes clear that for a given V, x approaches x_∞ exponentially fast.

Despite the underlying assumptions, both the Hodgkin Huxley and kinetic Markov models depicted above allow the same equations for their description, the ion-channel dynamics in a probabilistic fashion and the macroscopic ion-currents across the neuronal membrane in a deterministic fashion. In its current form, the model has been reduced enough to fit the SpiNNaker constraints. Modelling of the above equations in SpiNNaker allow the use of experimentally measurable and theoretically deducible parameters, which correspond to both macro-molecular and cellular scales. Such an approach enables the exploration of the effect of such scales on the high-dimensional dynamics of SNNs.

6.3.2 Voltage-Dependence of Transition Coefficients

Equation 6.14 gives the time evolution of the fraction of open channels, through the dynamics of activation and inactivation. For our results in section 6.4, We still need, an explicit expression for the transition coefficients. Following the transition state theory of reaction kinetics [TGK96], the starting point is to realise that different channel states, e.g., X_i and X_j, have in general different energies E_i and E_j [SQB99]. To transition between these two meta-stable configurations, the channel needs to overcome an energy barrier $\Delta E^*_{i,j} = E^*_{i,j} - E_i$ separating the two states. Here, $E^*_{i,j}$ is the energy of the transition state (the top of the potential energy barrier between states X_i and X_j) denoted by $*$. Beyond internal energy, the activation enthalpy adds pressure-volume

work: $\Delta H^*_{i,j} = \Delta E^*_{i,j} + \Delta vP$, similarly, the Gibbs energy of activation also takes entropy $\Delta S^* = k \ln (Z'_b/Z'_R)$ into account: $\Delta G^*_{i,j} = \Delta H^*_{i,j} - \Delta S^* T$. Here Z'_b and Z'_R denote the partition functions for the nonreactive degrees of freedom of the barrier and initial (reactant) states respectively. Hence, $\Delta G^*_{i,j} = G^*_{i,j} - G_i$ has to be acquired by the ion-channel segments in order to transition from the deactivated (inactivated) to the activated (deinactivated) states. It is this quantity which controls the transition rate, while $\Delta G_{i,j} = G_i - G_j$ determine the equilibrium distribution. The formulation in terms of energy allows a common approach in ion-channel modelling which is, to borrow the rate constants formulation from the theory of reaction kinetics for chemical reactions. Back in 1935, Henry Eyring [Eyr35a] [Eyr35b] already formulated the rate constant from the assumptions that the initial states achieve thermal quasi-equilibrium with the activated complex, an infinitesimal region around the transition state, from which the system transitions to the next state. This transition state theory (TST) [TGK96] finds:

$$r = \kappa \frac{k_B T}{h} e^{-\frac{\Delta G^*_{i,j}}{k_B T}}. \tag{6.15}$$

Where κ is a transmission coefficient which accounts for re-crossing of the energy barrier.

Computing 6.15 from first principles would require the exact form of the potential energy surface, which is very difficult to calculate for the proteins of interest here [RABI04, Sig14]. Still, equation 6.15 allows the inverse problem of deriving $\Delta H^*_{i,j}$, $\Delta S^*_{i,j}$ and $\Delta G^*_{i,j}$ if the transition rate is measured experimentally. Facing the difficulty in calculating $\Delta Gi, j^*$, one can express G_i as a Taylor series expansion over the membrane voltage, $G_i = A_i + B_i V + C_i V^2 + ...$, which gives rise to the so-called nonlinear thermodynamic model for the transition coefficients [DH00]. Hence, the standard energy of activation $\Delta Gi, j^* = a_{i,j} + b_{i,j} V + c_{i,j} V^2 + ...$, has a contribution $a_{i,j} = A^*_{i,j} - A_i$ from the free energy which is independent of the electric field $\vec{E}$ and a contribution $b_{i,j} = B^*_{i,j} - B_i$ from the interaction between $\vec{E}$ and independent charges or rigid dipoles, $c_{i,j} V^2$ on the other hand accounts for mechanical constraints due to the channel's structure, for electronic polarisation and for pressure induced by V, and so on for higher order terms in the expansion. Then,

$$r = \frac{k_B T}{h} e^{-\frac{a_{i,j} + b_{i,j} V + c_{i,j} V^2 + ...}{k_B T}}. \tag{6.16}$$

The number of terms from the expansion that would be needed can be determined by the fitting of equations 6.12, 6.13 and 6.14 to experimental data and using 6.16 with

varying orders. Furthermore, an explicit expression for the first order approximation can be found from $\Delta G = \Delta H - T\Delta S$, where both terms can be obtained experimentally varying T. It is also possible to consider that the channel opens through the displacement of an effective electric charge q_{eq} by an effective distance δ_{eq} where it reaches the activated complex. Hence, the fordward and backward terms are $\Delta H_f = \Delta H_f(V = 0) - \delta_{eq} z_{eq} FV$ and $\Delta H_b = \Delta H_b(V = 0) + (1 - \delta_{eq}) z_{eq} FV$ [SGGW11]. Where F is Faraday's constant. In this case we have:

$$r_f(V) = Ke^{\frac{-\delta(V - V_{1/2})}{\sigma}} \tag{6.17}$$

$$r_b(V) = Ke^{\frac{(1-\delta)(V - V_{1/2})}{\sigma}} \tag{6.18}$$

Where $r_f \in \{\alpha, \lambda\}$, $r_b \in \{\beta, \kappa\}$, $V_{1/2}$ is the half-activation voltage given by $V_{1/2} = zF\left[\Delta H_1^{(0)} - \Delta H_{-1}^{(0)} - T\left(\Delta S_1^{(0)} - \Delta S_{-1}^{(0)}\right)\right]$, σ is the inverse slope given by $\sigma = RT/z_{eq}F$ and K is the maximum rate parameter, which includes the voltage-independent energy components [BG91, DH00]:

$$K = \frac{k_B T}{h} e^{\frac{\delta_1 \Delta S_{-1}^{(0)} \delta_{-1} \Delta S_1^{(0)}}{R}} e^{\frac{\delta_1 \Delta H_{-1}^{(0)} \delta_{-1} \Delta H_1^{(0)}}{R}}. \tag{6.19}$$

With this simplification, we can then rewrite 6.12 as:

$$x_\infty = \frac{1}{1 + e^{-\frac{V - V_{1/2}}{\sigma}}} \tag{6.20}$$

and

$$\tau_x(V) = \frac{1}{Ke^{\frac{\delta(V - V_{1/2})}{\sigma}} + Ke^{\frac{-(1-\delta)(V - V_{1/2})}{\sigma}}}. \tag{6.21}$$

Here, δ controls the skew of the τ curve and both K, $V_{1/2}$ and σ can be determined experimentally. This more simplified formulation can be used to reduce hardware requirements when biological accuracy is less relevant than the computations performed by the neuron, this is usually the case for nonbiological problem-solving applications of neuromorphics [DSL+18, FGF17, HJM13].

A further step in detail can be introduced through the Kramers theory of reaction kinetics, an improvement to the transition state theory (TST), which besides assuming the existence of an activated transition state which is in quasi-equilibrium with the

initial states, here corresponding to the closed state, includes the presence of the background substances which may exchange energy with the ion-channel activated state, triggering its transition to either the open or the closed states. This is relevant because the state transitions are driven by the transmembrane electrical potential but are under the influence of thermal fluctuations and the viscosity of the extracellular medium. The Kramer's theory rest on the fundamental assumptions that the channel obeys detailed-balance and satisfies the fluctuation-dissipation theorem, in this way, Kramer's theory can be derived from the the generalised Langevin equation [TTBP91], allowing the formulation of a principled model in which the rate constant is given by:

$$r = \left(\frac{\omega_R \gamma}{4\pi\omega_B m}\right)\left[\left[1+\left(\frac{2\omega_B m}{\gamma}\right)^2\right]^{1/2} - 1\right] e^{-\frac{\Delta G^*_{i,j}}{k_B T}}. \tag{6.22}$$

Here, ω_R and ω_B define the curvature of a potential energy surface around the initial states and barrier respectively [SQB99]. γ is the extracellular medium viscosity, T is the temperature and k_B is the Boltzman constant. In the limiting case of very high viscosity equation 6.22 approximates to:

$$r = \frac{\omega_R \omega_B m}{2\pi\gamma} e^{-\frac{\Delta G^*_{i,j}}{k_B T}}. \tag{6.23}$$

Equation 6.23 opens the possibility of approximate first-principles modelling in SpiNNaker following 6.14, 6.12, 6.13, because ω_R, ω_B and γ can be obtained from molecular dynamics simulations.

Alternative Forms For The Rate Coefficients An alternative form for the time constant, which limits its value without the need for nonlinear terms in the rate coefficients and then simplifies implementation is given by adding a rate-limiting factor τ_0 [BG91].

$$\tau_x = \frac{1}{\alpha'(V) + \beta'(V)} + \tau_0. \tag{6.24}$$

This adjustment is very economical despite not being physically plausible [DH10].

Another important formulation is that used by software tools for fitting experimental data, these often use rate coefficients of the form

$$r(V) = \frac{A + BV}{C + H\, e^{\frac{V+D}{F}}} \tag{6.25}$$

To the author's knowledge, this form has not a clear derivation from physical

arguments, it increases the gap between Hodgkin-Huxley and Markov models and, by forcing an additional divide on τ_x and six parameters per coefficient, becomes exceedingly expensive for SpiNNaker 1. Nevertheless, implementing equation 6.25 for SpiNNaker 2, has the advantage of directly accepting input from data found in the literature and hence becomes worth to support.

6.3.3 From Ion-Channels to Membrane Conductance and Current

Currently, the neuron models in SpiNNaker assume a constant membrane conductance, an assumption that is true only in a very restricted region of biological neurons. Incorporating the channel dynamics described above allows the formulation of voltage- and time-dependent conductances, in turn, these are observed to be either ohmic or rectifying depending on the concentration difference of the participating ions across the cellular membrane. For a given ion A, if $[A]_{out}/[A]_{in} \approx 1$, the currents tend to behave linearly. If $[A]_{out}/[A]_{in} > 1$, as it is for Ca^{2+}, the ions will flow more easily from the outside to the inside of the neuron and an inward rectification is observed, similarly, outward rectification occurs when $[A]_{out}/[A]_{in} < 1$.

Consider the ensemble of channels of type A that may exist in the neuronal membrane at a given time, each contributing n VSDs and q inactivating segments, the evolution of the fraction of segments in the open state $[O] = m^n h^q$ (recall that all VSDs should be activated and the pore deinactivated in order for a channel to conduct) controls the in-flux and out-flux of ions to the cell, hence determining the instantaneous transmembrane conductance. If a single open channel causes a g_A increase in conductance, a membrane with channel density η_A reaches the maximum increase in conductance density $\hat{g}_A = \eta_A g_A$ when all channels are open. Then, the voltage-dependent gain in conductance density can be expressed as

$$g(V) = \hat{g}_A m(V)^n h(V)^q. \tag{6.26}$$

For Markov models with multiple conducting (open) states $g_A = \sum_i \hat{g}_{o_i} M_i$, with M_i the fraction of channels in the O_i open state. For the ion species for which the linearity condition is satisfied during ion electrodiffusion, equation 6.26 can be used along an ohmic approximation to the current density:

$$I_A = \hat{g}_A m^n h^q (V - E_A). \tag{6.27}$$

Here, E_A is the membrane potential at which no net flux of the ion A is observed. An explicit expression for E_A can be found by considering that when the channel is open, the ions move in aqueous media under the influence of drift and diffusion and in the absence of any energy barriers. Thus, the ion flux obeys the Nernst-Planck equation (NPE), $I_A = \mathcal{J}_A z_A F = -\left(\mu z^2 F[A]\frac{\partial V}{\partial x} + \mu z RT \frac{\partial [A]}{\partial x}\right)$, in which the first term describes the ion's interaction with the electric field and the second the influence of the concentration gradient, μ is the molar mobility of the ions and z their valence. By making $I_A = 0$ in the NPE, one obtains the Nernst equation $E_A = RT \ln([X]_{out}/[X]_{in})/z_x F$. This expression can be used to obtain E_A from experimentally measured concentrations, but also, to infer the effects of changes in those concentrations on the network as has been done by [TT14].

When rectification is present, as for example in Ca^{2+} currents, for which $[Ca^{2+}]_{out}/[Ca^{2+}]_{in} \approx 10^4$, the simplest nonlinear model for the current across the thickness of the membrane can be derived from the NPE by considering the steady-state ($\frac{dI}{dx} = 0$), noninteracting ions – so that these can be considered independent of each other– and that the electric field across the membrane with thickness l is constant, i.e., $\vec{E} = -\frac{dV}{dr} = \frac{\Delta V}{l}$. Under these conditions the Goldman-Hodgkin-Katz voltage- and concentration-dependent equation is obtained for the ion current [SGGW11]:

$$I_A = \mathcal{P}_A \frac{z_A^2 F^2 V}{RT}\left(\frac{[A]_{in} - [A]_{out}\, e^{-z_A FV/RT}}{1 - e^{-z_A FV/RT}}\right). \tag{6.28}$$

Where the membrane permeability to ion A, $\mathcal{P}_A = \frac{\mu RT\beta}{lF}$ has been derived from the Einstein relation for the diffusion coefficient $D = \frac{\mu RT}{F}$ and the assumption of a linear drop (or rise) in concentration across the membrane.

When the Nernst-Planck equation does not describe adequately the ion movement traversing the open channel, due to, for example, the presence of energy barriers, a description from the transition state theory similar to that exposed above for the rate coefficients of the channel conformations is necessary. Alternatively, Poisson-Nernst-Plank equations allow the inclusion of inhomogeneous (position-dependent) diffusion coefficients by using the Poisson equation to relate ion concentration and the dielectric response function across the channel [LE14, LE15]. This enters as a term in the free energy function of the NPE.

In contrast, fine-grained approaches such as Brownian dynamics or all-atom molecular dynamics seem out of the scope of any near-future network-level neuromorphic

computing architecture as even conventional computers struggle to simulate a time-interval long enough to observe a single ion crossing through the channel [FCAA10].

6.3.4 Temperature Dependence

It is common to adjust for the simulation temperature T_s given the temperature T_m at which the data was recorded. This is done by multiplying the maximum conductance and either the rate coefficients or the time constants by the experimentally accessible factor $Q_{10} = r_i(T+10)/r_i(T)$:

$$\hat{g}(T_s) \longleftarrow \hat{g}(T_m) Q_{10}^{\frac{T_s - Tm}{10}} \tag{6.29}$$

with

$$\alpha(T_s) \longleftarrow \alpha(T_m) Q_{10}^{\frac{T_s - Tm}{10}} \tag{6.30}$$

$$\beta(T_s) \longleftarrow \beta(T_m) Q_{10}^{\frac{T_s - Tm}{10}} \tag{6.31}$$

or

$$\tau_x(T_s) \longleftarrow \tau_x(T_m) Q_{10}^{\frac{T_s - Tm}{10}} \tag{6.32}$$

$$m_\infty(T_s) \longleftarrow m_\infty(T_m) \tag{6.33}$$

Tipical values for Q_{10} range between 10^0 and10^1 for both conductance [SGGW11] and rate constants [Foh15]. Hence, it is straightforward to include it in the implementation and provides the ability to change the simulated temperature across runs and infer its network-level effect. If the activation energy E_a has been deduced from theoretical considerations Q_{10} can be determined as $Q_{10} = e^{10E_a/[k_B T(T+10)]}$. Similarly, the activation energy can be obtained from the measured Q_{10}.

6.4 Implementation

In this section, we present our preliminary implementation of intrinsic currents on SpiNNaker based on experimental data [HT05]. Such data is conventionally acquired through voltage-clamp and patch-clamp experiments on biological neurons with various chemical baths [SN84].

6.4.1 Considerations for SpiNNaker

The hardware architecture of SpiNNaker imposes a few but important constraints into the software development cycle:

- Firstly, the whole neural model should fit into DTCM (32KB).
- Secondly, the error introduced by the fixed-point arithmetic should not change the qualitative behaviour of the model. In the sense of not being too sensitive to initial conditions.
- Thirdly, firing rates cannot saturate the communication fabric.
- Lastly, to run in realtime all state update operations need to always be completed within a millisecond time interval by each processor.

Saturation of the interconnect is not relevant for the present study as we are concerned with the local properties of the neurons by adding up to their dynamical complexity, SpiNNaker is well suited for the highest rate of $1\,spike/ms$ as long as there is enough spatial sparsity in the network. However, the other constraints above limit the implementability of the model. For realtime resolution, the state update operations that need to be completed within a millisecond for all neurons existing in a single processor include, the updating of the neuron membrane voltage equation (e.g. LIF or Izhikevich neuron models), the processing of incoming spikes, computation of plasticity rules (e.g. STDP, STD, STP), neuromodulation selectivity traces [RBB$^+$18], as well as the intrinsic currents considered here. Then, for certain current types the intrinsic currents overhead to the model, exposed in the previous section, demands the simulation be slowed down by a factor of 10 to keep their stability or accuracy. For this reason, the persistent sodium I_{NaP} and depolarization-activated potassium I_{DK} currents in figures 6.4, 6.5 and 6.6 were run with a timestep of $0.1\,ms$.

In the standard way of programming SpiNNaker, the software stack consists of a lower and higher-level C layers, a python layer which links the C models to the PyNN description language and a further python (or other high-level programming languages) application programming interface (API) for SNN-based algorithms [RBB$^+$18, RBD$^+$18]. For the simulations in this chapter, the fixed-point implementation of the exponential (*expk*) function developed by D. Lester [Les14] and available in the SpiNNaker software stack was implemented in the low-level C [Les14] (it follows a similar approach to the one in [PHE$^+$17]), the intrinsic currents and their integration with the neuron models

are implemented in high-level C and their object-oriented versions in python. These latter two implemented by the author and colleagues O. Rhodes and M. Mikaitis. The higher-level layers allow the integration of intrinsic currents with SpyNNaker, in this way intrinsic currents and voltage-clamp parameters can easily be modified or defined from the front end.

Due to the model complexity and requirements for precision, the widespread use of intrinsic currents may only happen for SpiNNaker 2. Two improvements from this next generation of SpiNNaker (see section 1.2) have a direct implication on the performance of intrinsic currents modelling. Firstly, the adoption of the ARM M4F cores implies access to a single-precision floating-point hardware unit [MHF19]. Such an extra precision improves both stability and accuracy for solving differential equations while maintaining a modest energy expenditure. In contrast, the SpiNNaker 1 hardware only supports fixed-point arithmetic (alongside the GCC implementation of the ISO 18037 standard [EWR09]), if any floating-point operation is needed, it has to be implemented in software, resulting in poorer performance and power efficiency. Secondly, and in contrast to the 95 cycle soft-exponential *expk*, SpiNNaker 2 is boosted by fixed-point (internal 39-bits with programmable approximation to 32-bit output) elementary function hardware accelerators, including exponential and logarithmic (base e) functions [MHF19, PHE$^+$17, MLS$^+$18]. The election of fixed-point arithmetic for the accelerators represents at least four times less area and energy than floating-point [Dal15]. When designing new neuromorphic hardware involving intrinsic currents, the arithmetic range of the parameters will dictate whether to support fixed- or floating-point architecture in hardware.

6.4.2 Intrinsic Currents Under Voltage Clamp in SpiNNaker

The intrinsic currents implementation is intended to follow the free dynamics of the membrane potential of each neuron. However, to verify the implementation it is important to compare the behaviour of these currents with that observed experimentally. As emphasized in section 6.1, each neuron contains thousands of ion channels spanning tens of current types, these currents are isolated experimentally by harnessing chemical blockers, intracellular and extracellular mediums and electrical properties. Once isolated, the dynamic response of the target current can be studied through a voltage clamp protocol. A voltage clamp consists in clamping the neuronal membrane to a certain hold potential V_h and stepping to a series of potentials $V_i \mid i \in \{1, 2, 3, ...\}$ after which V can be returned to V_h or to new clamp values. The objective is to measure the response

of the neuron to discrete changes in V. Such a response allows the extraction of $x_\infty(V)$ and $\tau_x(V)$. These are obtained for both activation and inactivation by following the maximum conductance, as well as the time course of the conductance when the neuron transitions from equilibrium at the hold potential to equilibrium at the step voltage. We can then use the parameterisation of $x_\infty(V)$ and $\tau_x(V)$ for the implementation of equation 6.14 in SpiNNaker. This in turn provides the evolution of m and h in equation 6.27, i.e., $I_A = \hat{g}_A m^n h^q (V - E_A)$. The activation and inactivation exponents n, q, as well as the equilibrium potential E_A are also obtained in these experiments.

Under voltage-clamp conditions:

$$x(t) = x_\infty(V_i) - (x_\infty(V_i) - x_0)\, e^{-t/\tau_x(V_i)} \tag{6.34}$$

Where x is either m or h. For simulation in SpiNNaker's discrete-time world, it is necessary to approximate the time evolution of x through small steps Δt, for tractability under SpiNNaker constraints, we assume the voltage as constant during such intervals. This is the same methodology used in SpiNNaker to integrate the LIF neuron model, which receives the current value of V at time t and returns an updated value for $t + \Delta t$. In a similar way, equation 6.34 defines the difference equation:

$$x_t(V_t) = x_\infty(V_t) + (x_{t-\Delta t}(V_{t-\Delta t}) - x_\infty(V_t))e^{-\Delta t/\tau_x(V_t)} \tag{6.35}$$

with x_∞ and τ_x determined experimentally or estimated with equations 6.12 and 6.13 alongside any of the techniques in section 6.3.2. The approximation of constant voltage across $1ms$ or $0.1ms$ intervals, will only be critical for rapidly varying currents or when using these as input into strongly nonlinear neuron models. Here, we follow [HT05] in using a modified LIF model where the intrinsic currents of table 6.1 enter as an addition to the input current to the membrane potential:

$$\frac{dV}{dt} = \frac{1}{\tau_m}\left[-g_{NaL}(V - E_{Na}) - g_{KL}(V - E_k) - I_{syn} - \sum_{int} I_{int} \right] - \frac{1}{\tau_{spike}} g_{spike}(V - E_k). \tag{6.36}$$

Where I_{int} corresponds to any of the intrinsic currents.

Current	Model Equations	
Dynamics: $\frac{dx}{dt} = \frac{x_\infty(V) - x}{\tau_x(V)}$ Update: $x_t = x_\infty(V_1) - (x_\infty(V_1) - x_{t-1})e^{\frac{-\Delta t}{\tau_x(V_1)}}$		
I_h	$m_\infty = \frac{1}{1+e^{(V-(-75))/5.5}}$	$\tau_m = \frac{1}{e^{(-14.59-0.086V)} + e^{(-1.87+0.0701V)}}$
I_T	$m_\infty = \frac{1}{1+e^{-(V+59.0)/6.2}}$ $h_\infty = \frac{1}{1+e^{(V+83)/4}}$	$\tau_m = \frac{0.22}{e^{-(V+132.0)/16.7}} + e^{(V+16.8)/18.2} + 0.13$ $\tau_h = \frac{8.2+56.6+0.27e^{(V+115.2)/5.0}}{1.0+e^{(V+86.0)/3.2}}$
$I_{Na(P)}$	$m_\infty = \frac{1}{1+e^{(-V+55.7)/7.7}}$	$\tau_m = 0$
Dynamics: $\frac{dD}{dt} = D_{influx} - \frac{D(1-D_{eq})}{\tau_D}$ Update: $D_t = (D_{influx} + \frac{D_{eq}}{\tau_D})(1 - e^{\frac{\Delta t}{\tau_D}}) + D_{t-1}e^{\frac{\Delta t}{\tau_D}}$ Update: $x_t = x_\infty(V_1)$		
I_{DK}	$m_\infty = \frac{1}{1+(d_{1/2}D^{3.5})}$	$D_{influx} = \frac{1}{1+e^{-(V-D_\theta)/\sigma_D}}$

Table 6.1: Model equations for intrinsic currents of relevance in the wakefulness-sleep transition (for further details, see [HT05] and references therein).

6.4.2.1 Simulation of Intrinsic Currents from the Thalamocortical System

We show in figures 6.4-6.6 the implementation in SpiNNaker of the intrinsic currents summarised in table 6.1. These currents are present in diverse neurons of the thalamocortical system and are relevant for both sleep regulation, as well as for the wakefulness-sleep transition [HT05]. We follow the parameterisation in [HT05] and references therein for the pacemaker cation current I_h, the persistent sodium current I_{NaP}, and the depolarization-activated potassium current I_{DK}. These have been obtained elsewhere from fittings to experimental electrophysiological recordings of neurons undergoing diverse voltage-clamp protocols [HM92, CSVMW03, FFG96, WL03]. Their modulation was found to contribute to the transitions between different sleep stages in the model of the thalamocortical system developed by S. Hill and G. Tononi [HT05]. I_h is a noninactivating hyperpolarisation-activated cation current. It has a pacemaker role and is present in thalamic and intrinsically bursting cells [HM92]. I_{NaP} is present in most cortical neurons, it presents a quick activation near the resting potential but follows a slow inactivation, this is the reason for $\tau_m = 0$ and no inactivation in table 6.1. The low-threshold fast-activating calcium current I_T, together with I_h, is responsible for bursting behaviour in thalamic relay cells [MB97]. The depolarization-activated potassium current I_{DK} is involved in the termination of the depolarized phase of the slow oscillation during sleep. Although it is activated by Na^+ or Ca^{2+}, these ions concentration is modeled implicitly through a dependence in a depolarisation measure D (table 6.1). This is because the rise in their concentrations is proportional to the rise in membrane potential (depolarization). The diverse dependence of the equations in table 6.1 and other represent a challenge for their implementation in SpiNNaker, although hardcoding each particular current is doable (as we did here), such a process would require expertise and access to the low-level software of the SpiNNaker software stack. Instead, we aim to give high-level general support for integration of currents derived experimentally or in a principled way. Nevertheless, the hardware imposes hard constraints on the functional forms that can be reliably supported, this was the main motivation for the theoretical exploration presented in previous sections and future work will follow from it.

Figures 6.2 and 6.3 show the implementation of the I_h and I_T currents in SpiNNaker (dashed lines) and CPU with double-precision floating-point arithmetic (dotted lines). The difference between in-hardware fixed-point and floating-point implementations is within 10 % and will be critical only when high-accuracy is desired in the correspondence between experimental and simulated spike times. For I_h A voltage-clamp protocol

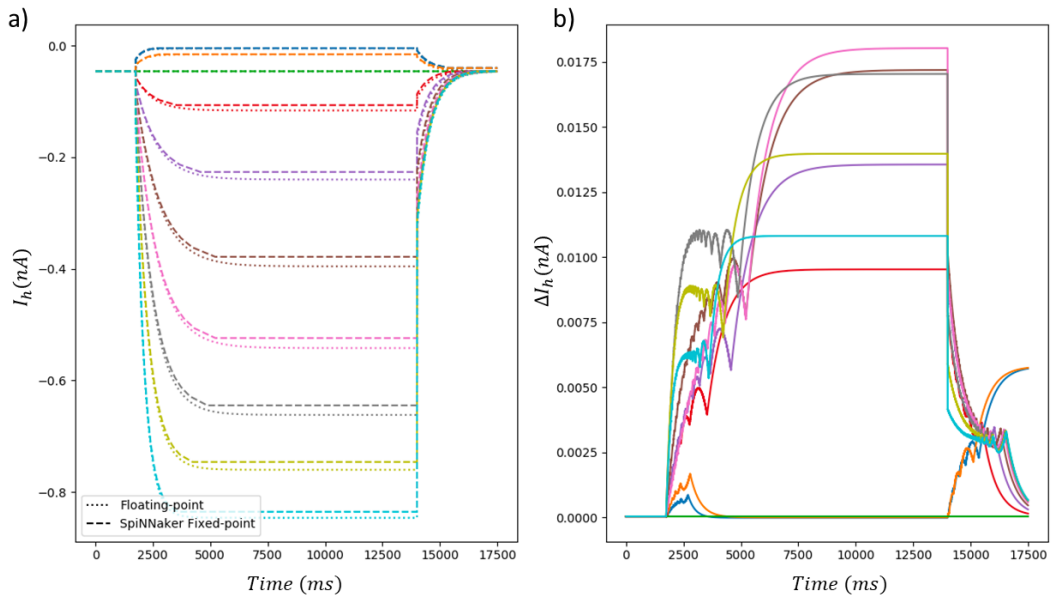


Figure 6.2: a) 1.0ms timestep implementation in SpiNNaker of the I_h current (dashed lines) compared with the python double-floating point implementation (dotted lines), both undergoing the same voltage clamp protocol. The neuron has been submitted to the same clamp protocol as that of figure 5 in [HM92]. b) Difference between SpiNNaker fixed-point and CPU floating-point implementations.

has been applied to the neuron according to the experimental results in [HM92]. The curves show good qualitative and quantitative correspondence with the experimental measures shown in figure 5 in [HM92].

Figures 6.4 and 6.5 show the SpiNNaker implementation of the I_{DK} and I_NaP, as well as their absolute and relative error when compared with a CPU implementation with full-precision double floating-point implementation. Here we used a $0.1ms$ simulation timestep, see in table 6.1 how the I_DK includes a fractional exponent which further affects precision. This causes that under a $1ms$ timestep the simulation suffers from overflowing and propagation of rounding errors in fixed-point arithmetic making the current to evaluate to zero for some clamp potentials. Yet, by sacrificing realtime performance the simulation can also generate results within an absolute error of 10% and a relative error of 1 %. In contrast, the simple formulation of the I_NaP current results in very good error margins in the order of 10^-3.

Finally, in figure 6.6 we compare our SpiNNaker implementation with an equivalent implementation on the state-of-the-art neural simulator *Nest*[1] [LLM+18]. For this, we

[1]Kindly provided by Andr Sevenius and Hans Ekkehard Plesser from the Institute of Basic Medical Sciences at Universitetet i Oslo.

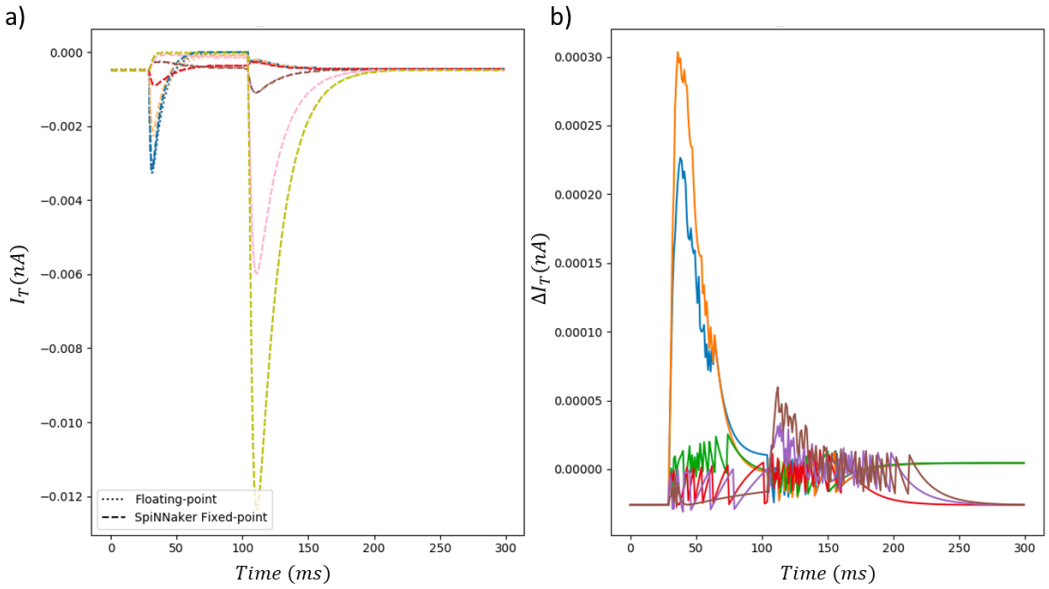


Figure 6.3: a) 1.0ms timestep implementation in SpiNNaker of the I_T current (dashed lines) compared with the python double-floating point implementation (dotted lines), both undergoing the same voltage clamp protocol. The neuron has been submitted to the same clamp protocol as that of figure 5 in [HM92]. b) Difference between SpiNNaker fixed-point and CPU floating-point implementations.

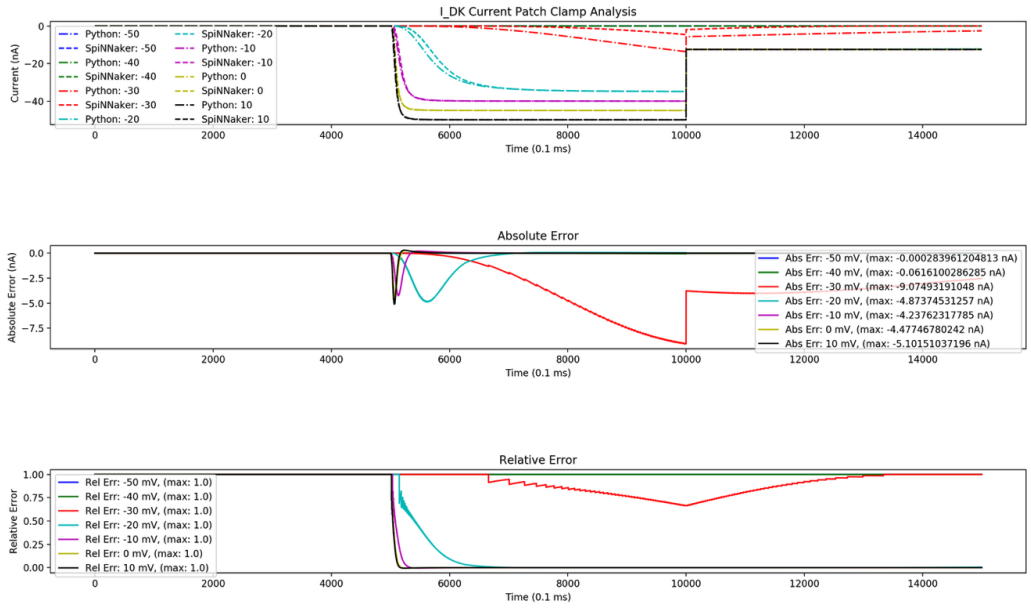


Figure 6.4: Implementation in SpiNNaker of the I_{DK} current compared with the python double-floating point implementation. A voltage clamp protocol has been applied to the neuron for various holding voltages.

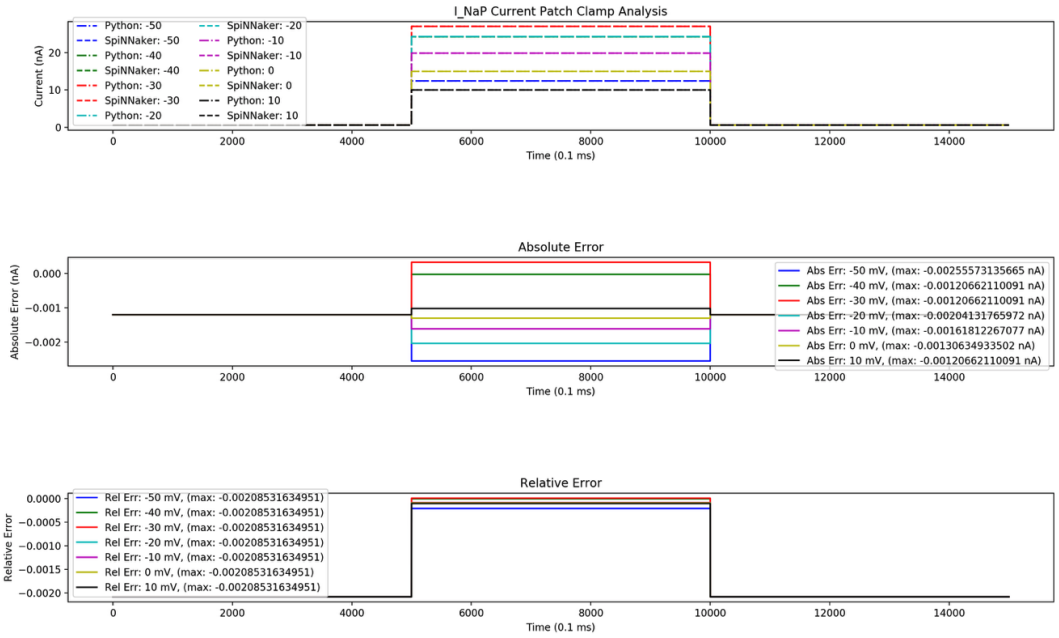


Figure 6.5: Implementation in SpiNNaker of the I_{NaP} current compared with a python double-floating point implementation. A voltage clamp protocol has been applied to the neuron for various holding voltages.

endowed a LIF neuron with the I_{DK} and I_{NaP} currents of figures 6.4 and 6.5. Figure 6.6 shows the behaviour of the membrane potential and both currents when the neuron is undergoing a current clamp. Although mismatch still exists in the exact spike times due to the difference in numerical precision, the neuron exhibits spike adaptation as expected and SpiNNaker matches exactly the numbers of spikes.

Author Contributions

The work presented in this chapter was done by the author (GAFG) in collaboration with Oliver Rhodes (OR) and Mantas Mikaitis (MM) for coding, figures and discussions. Professor Johan Storm, provided guidance and feedback throughout development (JFS). GAFG, OR, MM and JFS conceived and designed the study; GAFG and OR coded the implementation; MM profiled the performance of the scripts and traced low-level operations; GAFG, OR and MM interpreted and analysed the data; David Lester designed, coded and tested the *expk* function.

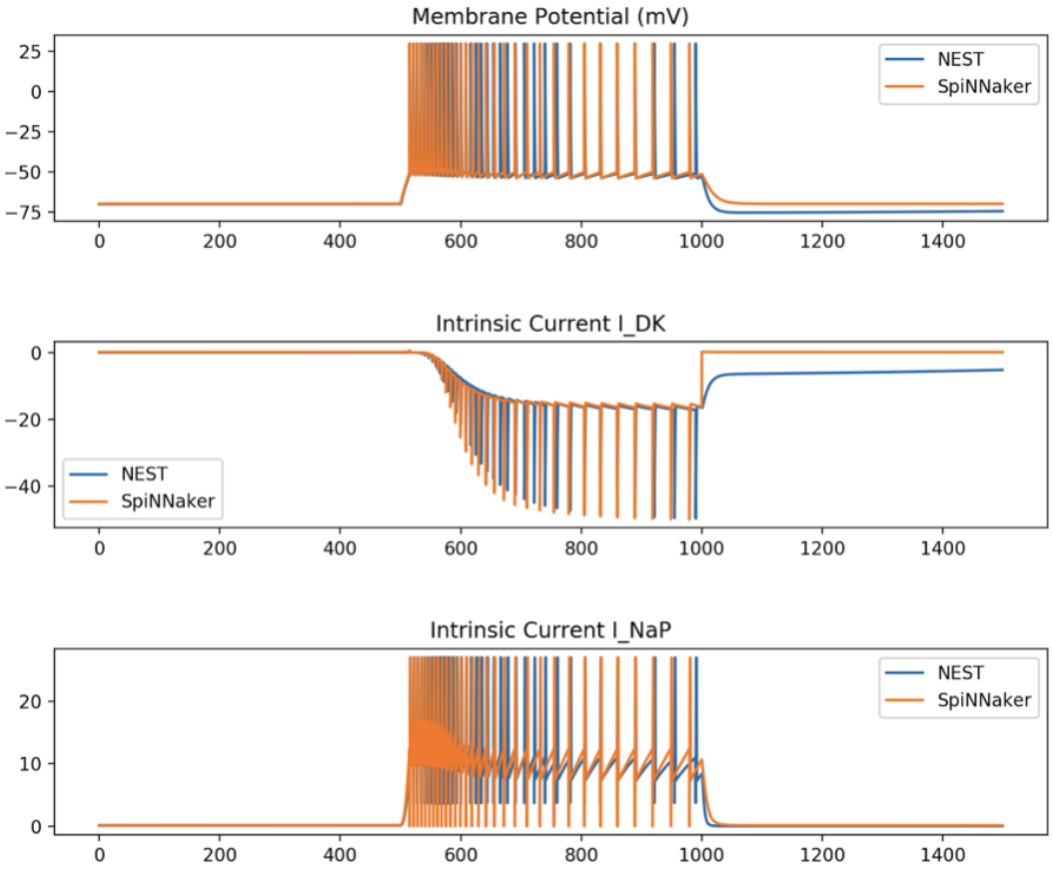


Figure 6.6: SpiNNaker implementation of spike adaptation (top) through the inclusion of the I_DK (middle) and I_NaP (down) intrinsic currents. The implementation using full-precision in Nest is shown for comparison. There is a mismatch between the spike times predicted by both systems but qualitative behaviour is achieved.

Acknowledgments

I am grateful to Nilsen Andr Sevenius for bringing the problem of modelling the thalamocortical system (and hence the need for implementing intrinsic currents) in the first place. I am also thankful to both Andr and Hans Ekkehard Plesser for sharing their Nest code and providing support.

Chapter 7

Conclusions and Future Work

In chapter 1 we stated the following hypothesis:

H1 Constraint satisfaction problems can be solved efficiently on neuromorphic hardware.

H2 Neuromorphic hardware can deliver competitive problem-solving with low energy expenditure and biologically relevant processing times.

H3 It is possible to validate CSP SNNs online (on-chip) without probing the neural activity of the entire network.

H4 Realistic postsynaptic currents are linearisable, allowing for their efficient implementation on neuromorphic hardware.

H5 Intrinsic currents are implementable in SpiNNaker

After the development of this thesis and the results presented in chapters 2, 4, 5 and 6, we can answer the hypothesis as follows:

H1&2 Chapters 2 and 4 demonstrate that CSP problems can indeed be solved in neuromorphic hardware with competitive performance in time while requiring a low energy consumption.

H3 We developed and implemented in chapter 4 a methodology for the on-chip validation of CSPs demonstrating its functionality. Hence, no probing or postprocessing is required for solving CSPs in Loihi.

H4 Hypothesis H4 has been proved in chapter 5 leading to the addition of the alpha and postsynaptic currents implementation into SpiNNaker's software stack.

H5 The preliminary results of chapter 6 show that voltage-gated ion-channel currents can be implemented in SpiNNaker and that these influence the neuron dynamics as expected. Qualitative behaviour is preserved while some quantitative discrepancies are observed. Thus, the adequacy of an implementation of intrinsic currents in SpiNNaker depends on whether qualitative or quantitative accuracy is required.

7.1 Constraint Satisfaction Solvers

In this thesis, we demonstrated the solution of constraint satisfaction problems (CSPs) using the SpiNNaker and Loihi neuromorphic chips. CSPs have the advantage of no requiring massive training datasets, while being useful in a wide variety of applications, from scheduling in factories to compiler optimisation. We used CSPs of distinct nature to highlight this flexibility and extent of applicability on our neuromorphic solvers.

Although a thorough benchmark is needed, it is possible to have at least one order of magnitude improvement over classical solvers. We showed performance ranging from milliseconds to tens of seconds. For comparison, complete traditional solvers, those guaranteed to find a solution if it exists, have typical solving times in the order of seconds to minutes. While the best general-purpose optimisation tools, which use both heuristics and constraint propagation, solve in milliseconds to seconds. Additionally, the D-wave quantum computer has been reported to find solutions in microseconds to seconds. Due to our time constraints, the results shown here have not been fully optimised to explore the resources of the chip maximally, this means that with further research, neuromorphics have the potential of becoming the state-of-the-art solutions for energy-efficient constraint satisfaction.

It is worth mentioning that in this thesis we did not focus on the important issue of scalability to very large problems. This was because the more immediate problems of demonstrating feasibility, convergence, competitive performance and on-chip validation had to be solved first. Larger problems in both SpiNNaker and Loihi are possible, but not explored here because our APIs were based on dense Numpy arrays for which memory requirements on the host increase rapidly. Such a limitation is easily overcome by refactoring the API to use Scipy sparse matrices. Also, due to the version of the sPyNNaker software used for the SpiNNaker solver (three years old at the time of writing) the compiler was not optimised and required hours or days for partitioning and resource allocation for large Sudoku problems, e.g. 16×16 up to 64×64. Since then, the compiler has been optimised and a refactoring of the API to the latest sPyNNaker

version should allow the handling of larger problems.

7.1.1 Key Contributions

- Development of an application programming interface (API) for the solution of CSPs in the massively parallel SpiNNaker machine. The API constitutes a stochastic general-problem incomplete solver which requires a few seconds to solve Sudoku and map colouring problems and converges to the problem solution for CSPs of moderate difficulty.

- Development of software and theoretical frameworks for the solution and on-chip validation of CSPs in Intel's Loihi neuromorphic chip and systems. The multicompartment spiking neural network which solves the problem also measure it's own cost function and notifies the host CPU when a solution to the problem is found. Only at this point, the solution is read from the network state. This methodology avoids constantly reading the network and transferring the data to host for post-processing, both of which cause an overhead of several minutes and seconds on SpiNNaker and Loihi respectively. On-chip validation implies satisfaction is known online. In our SpiNNaker solver, the solution and whether one was found is only known after gathering and processing the recorded network activity. The solver in Loihi achieves a speedup of two to three orders of magnitude improvement for Sudoku and map colouring problems (problems are solved in milliseconds).

All in all, we have demonstrated the usefulness of neuromorphic hardware for real-world problem-solving.

7.1.2 Future Work

Avenues for future work on constraint satisfaction are:

- *Scalability*: to allow scaling to larger CSP problems, the SpiNNaker and Loihi APIs should be refactored to use sparse matrices and the latest compiler versions. Due to the regularity and straightforward definition of these problems, it is also possible to modify the compilers to be more efficient in resources allocation for CSPs. We mentioned in chapter 4 how a hierarchy of integration neurons is necessary for adding the local contributions to the energy function in on-chip

validation. Testing and validation of such a methodology is also required for handling larger problems online.

- *Constraints:* the problems presented in this thesis use the *all-different* constraint type. A logical next step is to extend the solver to other constraint types, e.g. the inequality constraints for the travelling salesman problem in [JHM16].

- *Embedded systems*: one of the promising applications of our solvers is to use them on the edge in robotics and wearables.

- *Realtime*: our hypothesis that the solvers can interact with biological systems in realtime should be tested.

- *Heuristics:* our results from chapters 2 and 4 are a proof-of-concept and use the stochastic network dynamics to solve the problem. Further strategies can be borrowed from the state-of-the-art algorithms.

7.2 Postsynaptic Integration

The second part of the thesis enabled increased biological realism on the SpiNNaker machine. Chapter 5 demonstrates the implementation of biologically realist postsynaptic currents. The hard constraint is that required memory (buffer size and number of buffers) should not increase with the input spikes history. Thus, all previous activity has to be encoded on the last value in the memory local to the neuron, otherwise, the simulation will not be sustainable.

7.2.1 Key Contributions

- Development of a theoretical framework for the implementation in SpiNNaker of more biologically plausible alpha- and beta-shaped postsynaptic currents. These currents were subsequently implemented and integrated into the SpiNNaker software stack. Previously, SpiNNaker only supported Dirac delta and exponential kernels. These shapes control how synaptic input is integrated by a neuron into its membrane potential, directly affecting its spiking behaviour.

7.2.2 Future Work

- *Exact integration*: we mentioned in chapter 5 how our derivation is a subset of the more general framework of [RD99]. This latter includes all time-invariant linear systems. In particular, systems involving exponentially damped, oscillatory and polynomial dynamics. Further research can leverage these systems for dynamical systems simulations in neuromorphic hardware. Here, each SpiNNaker core would solve a set of time-invariant linear systems through exact integration and perform asynchronous event-driven updates according to a predefined set of rules.

- *Plasticity*: several learning algorithms for SNNs relay on plasticity rules for updating the synaptic weights. In SpiNNaker, plasticity rules like STDP require the use of exponential traces to save the presynaptic and postsynaptic spike history, it is worth exploring the use of alpha, beta and sinusoidal traces, for example, for the credit-assignment type of problems in reinforcement learning.

7.3 Voltage-gated Ion-channel Currents

Intrinsic currents originating from the collective behaviour of voltage-gated ion-channels underlay the complex changes in both, firing patterns and subthreshold dynamics observed in real neurons. These are the basis for homeostasis, bursting and intrinsic plasticity. In chapter 6 we demonstrated how these currents can be integrated into the SpiNNAker machine to achieve adaptive dynamics in spiking neurons which depends on the network activity. This means that the neurons will respond with a repertoire of firing behaviours depending on the current types these integrate, as well as on the network dynamics. This is in contrast to neuron models which require parameter tuning to change a neuron's response. Furthermore, our implementation paves the way for the use of experimental recordings from neurophysiology and bring neuromorphics closer to experimental neuroscience.

7.3.1 Key Contributions

- SpiNNaker implementation of voltage-gated ion-channel currents, also known as intrinsic currents. These follow the Hodgkin-Huxley formalism in which the neuron conductance is formulated in terms of the ensemble response of thousands of ion-channels across the neuronal membrane. The integration of data

from electrophysiological recordings e.g., from voltage-clamp experiments on biological neurons, as well as their effect on the neuron dynamics, e.g., spike adaptation, is achieved. The intrinsic currents response to voltage changes remains within 10 % of the CPU double floating-point version, this error is expected due to the use of fixed-point arithmetic. Spike adaptation has qualitative correspondence with an equivalent implementation on the state-of-the-art neural simulator Nest. There are however quantitative differences on the exact spike times, all within a few milliseconds, yet the same number of spikes is observed.

7.3.2 Future Work

- *Experimental Data*: to obtain the results shown in chapter 6 we hardcoded each current type across the different layers of the software stack. Nevertheless, we have exposed in chapter 6 the theoretical background for supporting generic current types. The next step is to give support to a generic current which is parameterised with experimental fittings to represent the various current types. Thus allowing the integration of experimental data from the front-end.

- *Analysis*: once arbitrary currents can be coded from the sPyNNaker front end, exhaustive comparison of the currents available in the Channelpedia [RKG+11] and voltage-clamp results from SpiNNaker will reveal the usefulness of SpiNNaker1 for simulating biological networks with increased biological detail.

- *SpiNNaker2*: given the encouraging results of chapter 6, the models can be ported to SpiNNaker2 while in the prototyping stage to influence further hardware design decisions.

- *Arithmetics*: some issues have been found on the GCC implementation of the ISO 18037:2008 standard fixed-point arithmetic [Mik20], as well as improvements for the numerical accuracy of operations with SpiNNaker fixed-point arithmetic [HMLF19]. It is important to assess how such results can improve the intrinsic currents implementation and lower their error.

Bibliography

[Abe91] M. Abeles. *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press, first edition, February 1991.

[ACG+09] Frederico A.C. Azevedo, Ludmila R.B. Carvalho, Lea T. Grinberg, Jos Marcelo Farfel, Renata E.L. Ferretti, Renata E.P. Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, 513(5):532–541, 2009.

[ACM12] K. Abuhassan, D. Coyle, and L. P. Maguire. Investigating the neural correlates of pathological cortical networks in alzheimer's disease using heterogeneous neuronal models. *IEEE Transactions on Biomedical Engineering*, 59(3):890–896, March 2012.

[ADGV15] Greg Aloupis, Erik D Demaine, Alan Guo, and Giovanni Viglietta. Classic nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

[AH89] Kenneth I Appel and Wolfgang Haken. *Every planar map is four colorable*, volume 98. American Mathematical Society Providence, 1989.

[ALS82] Daniel L Alkon, Izja Lederhendler, and Jonathan J Shoukimas. Primary changes of membrane currents during retention of associative learning. *Science*, 215(4533):693–695, 1982.

[ARTB19] Damiano Azzalini, Ignacio Rebollo, and Catherine Tallon-Baudry. Visceral signals shape brain dynamics and cognition. *Trends in cognitive sciences*, 2019.

[ARTD00] S Armano, P Rossi, V Taglietti, and Egidio D'Angelo. Long-term potentiation of intrinsic excitability at the mossy fiber–granule cell synapse of rat cerebellum. *Journal of Neuroscience*, 20(14):5208–5216, 2000.

[AS15] Mohammed Riyaz Ahmed and BK Sujatha. A review on methods, issues and challenges in neuromorphic engineering. In *Communications and Signal Processing (ICCSP), 2015 International Conference on*, pages 0899–0903. IEEE, 2015.

[Bar82] Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[BCHE19] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, page 1. ACM, 2019.

[Bek02] John M Bekkers. Synaptic transmission: a new kind of inhibition. *Current biology*, 12(19):R648–R650, 2002.

[BG91] LYLE J Borg-Graham. Modeling the non-linear conductances of excitable membranes. *Cellular Neurobiology: A Practical Approach*, 13:247–275, 1991.

[BGM+14] B. V. Benjamin, P. 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.

[BIP16] Jonathan Binas, Giacomo Indiveri, and Michael Pfeiffer. Spiking analog vlsi neuron assemblies as constraint satisfaction problem solvers. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2094–2097. IEEE, 2016.

[BLR+02] Björn Brembs, Fred D Lorenzetti, Fredy D Reyes, Douglas A Baxter, and John H Byrne. Operant reward learning in aplysia: neuronal correlates and mechanisms. *Science*, 296(5573):1706–1709, 2002.

[BM19] Hrishav Bakul Barua and Kartick Chandra Mondal. Approximate computing: A survey of recent trendsbringing greenness to computing and communication. *Journal of The Institution of Engineers (India): Series B*, pages 1–8, 2019.

[BS91] J M Bekkers and C F Stevens. Excitatory and inhibitory autaptic currents in isolated hippocampal neurons maintained in cell culture. *Proceedings of the National Academy of Sciences*, 88(17):7834–7838, 1991.

[BS09] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186, 2009.

[BS12] Ed Bullmore and Olaf Sporns. The economy of brain network organization. *Nature Reviews Neuroscience*, apr 2012.

[BW80] John F Brons and Charles D Woody. Long-term changes in excitability of cortical neurons after pavlovian conditioning and extinction. *Journal of Neurophysiology*, 44(3):605–615, 1980.

[BW12] György Buzsáki and Brendon O Watson. Brain rhythms and neural syntax: implications for efficient coding of cognitive content and neuropsychiatric disease. *Dialogues in clinical neuroscience*, 14(4):345, 2012.

[BZ08] Ivo Blöchliger and Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, 2008.

[CAW16] Ying Chen, JD Elenee Argentinis, and Griff Weber. Ibm watson: how cognitive computing can be applied to big data challenges in life sciences research. *Clinical therapeutics*, 38(4):688–701, 2016.

[CB41a] Kenneth S Cole and Richard F Baker. Longitudinal impedance of the squid giant axon. *The Journal of general physiology*, 24(6):771–788, 1941.

[CB41b] Kenneth S Cole and Richard F Baker. Transverse impedance of the squid giant axon during current flow. *The Journal of general physiology*, 24(4):535–549, 1941.

[CB90] Michael C Crair and William Bialek. Non-boltzmann dynamics in networks of spiking neurons. In *Advances in neural information processing systems*, pages 109–116, 1990.

[CBB+18] Scott Cyphers, Arjun K. Bansal, Anahita Bhiwandiwalla, Jayaram Bobba, Matthew Brookhart, Avijit Chakraborty, William Constable, Christian Convey, Leona Cook, Omar Kanawi, Robert Kimball, Jason Knight, Nikolay Korovaiko, Varun Kumar Vijay, Yixing Lao, Christopher R. Lishka, Jaikrishnan Menon, Jennifer Myers, Sandeep Aswath Narayana, Adam Procter, and Tristan J. Webb. Intel ngraph: An intermediate representation, compiler, and executor for deep learning. *CoRR*, abs/1801.08058, 2018.

[CBBA14] Martin Cerny, Cyril Brom, Roman Barták, and Martin Antos. Spice it up! enriching open world npc simulation using constraint satisfaction. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[CD06] Robert C Cannon and Giampaolo D'Alessandro. The ion channel inverse problem: neuroinformatics meets biophysics. *PLoS computational biology*, 2(8):e91, 2006.

[CFT+15] Aaron Michael Clarke, Johannes Friedrich, Elisa M. Tartaglia, Silvia Marchesotti, Walter Senn, and Michael H. Herzog. Human and machine learning in non-markovian decision making. *PLOS ONE*, 10(4):1–15, 04 2015.

[Cha82] G. J. Chaitin. Register allocation & spilling via graph coloring. *SIGPLAN Not.*, 17(6):98–101, June 1982.

[CHDW87] M Chams, A Hertz, and D De Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, 1987.

[Chu08] Patricia Smith Churchland. The impact of neuroscience on philosophy. *Neuron*, 60(3):409 – 411, 2008.

[CKA+17] Andrew Currin, Konstantin Korovin, Maria Ababi, Katherine Roper,

Douglas B. Kell, Philip J. Day, and Ross D. King. Computing exponentially faster: implementing a non-deterministic universal turing machine using dna. *Journal of The Royal Society Interface*, 14(128), 2017.

[CMP+16] R Carter, J Mazurier, L Pirro, JU Sachse, P Baars, J Faul, C Grass, G Grasshoff, P Javorka, T Kammler, et al. 22nm fdsoi technology for emerging mobile, internet-of-things, and rf applications. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 2–2. IEEE, 2016.

[Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*, pages 24–30. North-Holland Publishing, 1965.

[Col84] Charles J. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8(1):25 – 30, 1984.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[CPC16] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[CRO+19] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews*, 119(19):10856–10915, 2019.

[CS+02] Marco Chiarandini, Thomas Stützle, et al. An application of iterated local search to graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, 2002.

[CSVMW03] Albert Compte, Maria V Sanchez-Vives, David A McCormick, and Xiao-Jing Wang. Cellular and network mechanisms of slow oscillatory activity (¡ 1 hz) and wave propagations in a cortical network model. *Journal of neurophysiology*, 89(5):2707–2725, 2003.

[CT04] Robert H Cudmore and Gina G Turrigiano. Long-term potentiation of intrinsic excitability in lv visual cortical neurons. *Journal of neurophysiology*, 92(1):341–348, 2004.

[DA+03] Peter Dayan, L Abbott, et al. Theoretical neuroscience: computational and mathematical modeling of neural systems. *Journal of Cognitive Neuroscience*, 15(1):154–155, 2003.

[Dai80] David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are np-complete. *Discrete Mathematics*, 30(3):289 – 293, 1980.

[Dal15] William Dally. High-performance hardware for machine learning. *NIPS Tutorial*, 2015.

[Dav08] Andrew P Davison. PyNN: a common interface for neuronal network simulators. *Front. Neuroinform.*, 2, 2008.

[Dav19] Mike Davies. Benchmarks for progress in neuromorphic computing. *Nature Machine Intelligence*, 1(9):386–388, 2019.

[DBS93] Alain Destexhe, Agnessa Babloyantz, and Terrence J Sejnowski. Ionic mechanisms for intrinsic slow oscillations in thalamic relay neurons. *Biophysical journal*, 65(4):1538–1552, 1993.

[DC+03] Rina Dechter, David Cohen, et al. *Constraint processing*. Morgan Kaufmann, 2003.

[DH99] Raphaël Dorne and Jin-Kao Hao. Tabu search for graph coloring, t-colorings and set t-colorings. In *Meta-heuristics*, pages 77–92. Springer, 1999.

[DH00] Alain Destexhe and John R Huguenard. Nonlinear thermodynamic models of voltage-dependent currents. *Journal of computational neuroscience*, 9(3):259–270, 2000.

[DH10] Alain Destexhe and John R Huguenard. Modeling voltage-dependent channels. *Computational modeling methods for neuroscientists. MIT Press, Cambridge*, pages 107–137, 2010.

[DIPR07] Kenji Doya, Shin Ishii, Alexandre Pouget, and Rajesh PN Rao. *Bayesian brain: Probabilistic approaches to neural coding*. MIT press, 2007.

[DM04] Rodney J Douglas and Kevan AC Martin. Neuronal circuits of the neocortex. *Annu. Rev. Neurosci.*, 27:419–451, 2004.

[DMD+96] D. Dupeyron, S. Le Masson, Y. Deval, G. Le Masson, and J. . Dom. A bicmos implementation of the hodgkin-huxley formalism. In *Proceedings of Fifth International Conference on Microelectronics for Neural Networks*, pages 311–316, Feb 1996.

[DNUH98] Alain Destexhe, Mike Neubig, Daniel Ulrich, and John Huguenard. Dendritic low-threshold calcium currents in thalamic relay cells. *Journal of Neuroscience*, 18(10):3574–3588, 1998.

[Don12] Zhou Yun Dong. Zsolver, 2012.

[DRT99] Niraj S Desai, Lana C Rutherford, and Gina G Turrigiano. Plasticity in the intrinsic excitability of cortical pyramidal neurons. *Nature neuroscience*, 2(6):515, 1999.

[DSL+18] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, January 2018.

[EA75] S F Edwards and P W Anderson. Theory of spin glasses. *Journal of Physics F: Metal Physics*, 5(5):965, 1975.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449467, 1965.

[Eli18] C. Eliasmith. Intel 2018 demo video, 10 April 2018.

[ERT12] Mária Ercsey-Ravasz and Zoltán Toroczkai. The chaos within sudoku. *Scientific Reports*, 2(1), oct 2012.

[ESC+12] 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.

[EWR09] Wilfried Elmenreich, Andreas Wolf, and Maximilian Rosenblattl. Providing standardized fixed-point arithmetics for embedded c programs. In *Intelligent Technical Systems*, pages 101–114. Springer, 2009.

[Eyr35a] Henry Eyring. The activated complex and the absolute rate of chemical reactions. *Chemical Reviews*, 17(1):65–77, 1935.

[Eyr35b] Henry Eyring. The activated complex in chemical reactions. *The Journal of Chemical Physics*, 3(2):107–115, 1935.

[FCAA10] Jordi Faraudo, Carles Calero, and Marcel Aguilella-Arzo. Ionic partition and transport in multi-ionic channels: a molecular dynamics simulation study of the ompf bacterial porin. *Biophysical journal*, 99(7):2107–2115, 2010.

[FFG96] Ilya A Fleidervish, A Friedman, and MJ Gutnick. Slow inactivation of na+ current and slow cumulative spike adaptation in mouse and guinea-pig neocortical neurones in slices. *The Journal of physiology*, 493(1):83–97, 1996.

[FFR+16] Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, Giovanni Pezzulo, et al. Active inference and learning. *Neuroscience & Biobehavioral Reviews*, 68:862–879, 2016.

[FGF17] Gabriel A. Fonseca Guerra and Steve B. Furber. Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Frontiers in Neuroscience*, 11:714, 2017.

[FGTP14] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665, May 2014.

[FJ05a] Bertram Felgenhauer and Frazer Jarvis. Enumerating possible sudoku grids. *Preprint available at http://www. afjarvis. staff. shef. ac. uk/sudoku/sudoku. pdf*, 2005.

[FJ05b] Andreas Frick and Daniel Johnston. Plasticity of dendritic excitability. *Journal of neurobiology*, 64(1):100–115, 2005.

[Flo05] R. V. Florian. A reinforcement learning algorithm for spiking neural networks. In *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*, pages 8 pp.–, Sept 2005.

[Flo12] Răzvan V Florian. The chronotron: a neuron that learns to fire temporally precise spike patterns. *PloS one*, 7(8):e40233, 2012.

[FLP+13] 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 Transactions on Computers*, 62(12):2454–2467, 2013.

[FLS01] Dimitris Fotakis, Spiridon Likothanassis, and Stamatis Stefanakos. An evolutionary annealing approach to graph coloring. *Applications of Evolutionary Computing*, pages 120–129, 2001.

[FMJ04] Andreas Frick, Jeffrey Magee, and Daniel Johnston. Ltp is accompanied by an enhanced local excitability of pyramidal neuron dendrites. *Nature neuroscience*, 7(2):126, 2004.

[FOF+20] E. Paxon Frady, Garrick Orchard, David Florey, Nabil Imam, Ruokun Liu, Joyesh Mishra, Jonathan Tse, Andreas Wild, Friedrich T. Sommer, and Mike Davies. Neuromorphic nearest-neighbor search using intel's pohoiki springs, 2020.

[Foh15] Jürgen F Fohlmeister. Voltage gating by molecular subunits of na+ and k+ ion channels: higher-dimensional cubic kinetics, rate constants, and temperature. *Journal of neurophysiology*, 113(10):3759–3777, 2015.

[For09] Lance Fortnow. The status of the p versus np problem. *Communications of the ACM*, 52(9):78–86, 2009.

[Fri05] Karl Friston. A theory of cortical responses. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456):815–836, 2005.

[Fri10] Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010.

[FS19] E Paxon Frady and Friedrich T Sommer. Robust computation with rhythmic spike patterns. *arXiv preprint arXiv:1901.07718*, 2019.

[FSF+13] Karl Friston, Philipp Schwartenbeck, Thomas FitzGerald, Michael Moutoussis, Tim Behrens, and Raymond J Dolan. The anatomy of choice: active inference and agency. *Frontiers in human neuroscience*, 7:598, 2013.

[Fur12] Steve Furber. To build a brain. *IEEE Spectr.*, 49(8):44–49, aug 2012.

[Fur16a] Steve Furber. Large-scale neuromorphic computing systems. *Journal of neural engineering*, 13(5):051001, 2016.

[Fur16b] Steve B Furber. Brain-inspired computing. *IET Computers & Digital Techniques*, 10(6):299–305, 2016.

[FWW+18] Huawei Fan, Yafeng Wang, Hengtong Wang, Ying-Cheng Lai, and Xingang Wang. Autapses promote synchronization in neuronal networks. *Scientific reports*, 8(1):580, 2018.

[GAD+13] Jared L Gearhart, Kristin L Adair, Richard J Detry, Justin D Durfee, Katherine A Jones, and Nathaniel Martin. Comparison of open-source linear programming solvers. *Sandia National Laboratories, SAND2013-8847*, 2013.

[GF11] Martin Grymel and Steve B. Furber. A novel programmable parallel CRC circuit. *IEEE Transactions on Very Large Scale, Integration (VLSI) Systems*, 19(10):1898–1902, oct 2011.

[GFS+19] Yury Gorbachev, Mikhail Fedorov, Iliya Slavutin, Artyom Tugarev, Marat Fatekhov, and Yaroslav Tarkan. Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[GH06] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, 2006.

[GJ79] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[GKK+13] Daniel Goodman, Behram Khan, Salman Khan, Mikel Luján, and Ian Watson. Software transactional memories for scala. *Journal of Parallel and Distributed Computing*, 73(2):150–163, feb 2013.

[GKNP14] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[GLH93] Bah-Hwee Gwee, Meng-Hiot Lim, and Jiun-Sien Ho. Solving four-colouring map problem using genetic algorithm. In *Artificial Neural Networks and Expert Systems, 1993. Proceedings., First New Zealand International Two-Stream Conference on*, pages 332–333. IEEE, 1993.

[GSMG07] Irene S Gabashvili, Bernd HA Sokolowski, Cynthia C Morton, and Anne BS Giersch. Ion channel gene expression in the inner ear. *Journal of the Association for Research in Otolaryngology*, 8(3):305–328, 2007.

[GWTH10] Fangzhen Ge, Zhen Wei, Yiming Tian, and Zhenjin Huang. Chaotic ant swarm for graph coloring. In *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, volume 1, pages 512–516. IEEE, 2010.

[H+01] Bertil Hille et al. *Ion channels of excitable membranes*, volume 507. Sinauer Sunderland, MA, 2001.

[Har04] David Harel. *Computers Ltd: What They Really Can't Do*. Oxford University Press, 2004.

[HB07] Kai M Hynna and Kwabena Boahen. Thermodynamically equivalent silicon models of voltage-dependent ion channels. *Neural Computation*, 19(2):327–350, 2007.

[Hea13] Corporate Headquarters. Programming with D-Wave: Map coloring problem. Technical report, D-Wave Systems, Inc., 2013.

[HF04] David Harel and Yishai A Feldman. *Algorithmics: the spirit of computing*. Pearson Education, 2004.

[HH52] Alan L Hodgkin and Andrew 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.

[HH09] Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3:31, 2009.

[Hig12] Rob High. The era of cognitive systems: An inside look at ibm watson and how it works. *IBM Corporation, Redbooks*, 2012.

[HJM13] Stefan Habenschuss, Zeno Jonke, and Wolfgang Maass. Stochastic computations in cortical microcircuit models. *PLOS Computational Biology*, 9(11):1–28, 11 2013.

[HK04] Christoph S. Herrmann and Andreas Klaus. Autapse turns neuron into oscillator. *International Journal of Bifurcation and Chaos*, 14(02):623–633, 2004.

[HM92] John R Huguenard and David A McCormick. Simulation of the currents involved in rhythmic oscillations in thalamic relay neurons. *Journal of neurophysiology*, 68(4):1373–1383, 1992.

[HMLF19] Michael Hopkins, Mantas Mikaitis, D Lester, and Steve Furber. Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural odes. *arXiv preprint arXiv:1904.11263*, 2019.

[HMU06] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Automata theory, languages, and computation. *International Edition*, 24, 2006.

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[HPZ08] Alain Hertz, Matthieu Plumettaz, and Nicolas Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

[HR97] Michael Häusser and Arnd Roth. Estimating the time course of the excitatory synaptic conductance in neocortical pyramidal cells using a novel voltage jump method. *Journal of Neuroscience*, 17(20):7606–7625, 1997.

[HS83] Geoffrey E Hinton and Terrence J Sejnowski. Optimal perceptual inference. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 448–453. IEEE New York, 1983.

[HSE+12] Sebastian Höppner, Chenming Shao, Holger Eisenreich, Georg Ellguth, Mario Ander, and René Schüffny. A power management architecture for fast per-core dvfs in heterogeneous mpsocs. In *2012 IEEE International Symposium on Circuits and Systems*, pages 261–264. IEEE, 2012.

[HT85] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, Jul 1985.

[HT05] Sean Hill and Giulio Tononi. Modeling sleep and wakefulness in the thalamocortical system. *Journal of neurophysiology*, 93(3):1671–1698, 2005.

[HVS02] Hua Hu, Koen Vervaeke, and Johan F Storm. Two forms of electrical resonance at theta frequencies, generated by m-current, h-current and persistent na+ current in rat hippocampal pyramidal cells. *The Journal of physiology*, 545(3):783–805, 2002.

[HW77] D. H. Hubel and T. N. Wiesel. Ferrier lecture: Functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society of London B: Biological Sciences*, 198(1130):1–59, 1977.

[HYV+17] S. Hppner, Y. Yan, B. Vogginger, A. Dixius, J. Partzsch, F. Neumrker, S. Hartmann, S. Schiefer, S. Scholze, G. Ellguth, L. Cederstroem, M. Eberlein, C. Mayr, S. Temple, L. Plana, J. Garside, S. Davison, D. R. Lester, and S. Furber. Dynamic voltage and frequency scaling for neuromorphic many-core systems. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017.

[IE08] Eugene M. Izhikevich and Gerald M. Edelman. Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Sciences*, 105(9):3593–3598, 2008.

[Izh03] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[JHM16] Zeno Jonke, Stefan Habenschuss, and Wolfgang Maass. Solving constraint satisfaction problems with networks of spiking neurons. *Frontiers in Neuroscience*, 10:118, 2016.

[Jon14] Zeno Jonke. *Stochastic Computations and Learning in Networks of Spiking Neurons: Simulation framework, Analysis and Theory*. PhD thesis, Institute for Theoretical Computer Science Graz University of Technology, January 2014.

[Jos20a] Ameet V. Joshi. *Amazon's Machine Learning Toolkit: Sagemaker*, pages 233–243. Springer International Publishing, Cham, 2020.

[Jos20b] Ameet V Joshi. Azure machine learning. In *Machine Learning and Artificial Intelligence*, pages 207–220. Springer, 2020.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[KD18] Nikolaus Kriegeskorte and Pamela K Douglas. Cognitive computational neuroscience. *Nature neuroscience*, 21(9):1148–1160, 2018.

[KLP+08] Muhammad Mukaram Khan, David R Lester, Luis A Plana, A Rast, Xin Jin, Eustace Painkras, and Stephen B Furber. Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2849–2856. Ieee, 2008.

[KM70] Bernhard Katz and R Miledi. Membrane noise produced by acetylcholine. *Nature*, 226(5249):962, 1970.

[KM18] Alexander Kugele and Karlheinz Meier. *Solving the Constraint Satisfaction Problem Sudoku on Neuromorphic Hardware*. Master Thesis, University of Heidelberg, 2018.

[KSE+14] Susanne Kunkel, Maximilian Schmidt, Jochen M Eppler, Hans E Plesser, Gen Masumoto, Jun Igarashi, Shin Ishii, Tomoki Fukai, Abigail Morrison, Markus Diesmann, et al. Spiking network simulation code for petascale computers. *Frontiers in neuroinformatics*, 8:78, 2014.

[KTK+16a] James C. Knight, Philip J. Tully, Bernhard A. Kaplan, Anders Lansner, and Steve B. Furber. Large-scale simulations of plastic neural networks on neuromorphic hardware. *Front. Neuroanat.*, 10, apr 2016.

[KTK+16b] James C Knight, Philip J Tully, Bernhard A Kaplan, Anders Lansner, and Steve B Furber. Large-scale simulations of plastic neural networks on neuromorphic hardware. *Frontiers in neuroanatomy*, 10:37, 2016.

[Kug18] Alexander Kugele. Solving the constraint satisfaction problem sudoku on neuromorphic hardware. Master's thesis, Kirchhoff Institute for Physics Heidelberg University, February 2018.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015.

[LDI+14] Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas. *Event-based neuromorphic systems*. John Wiley & Sons, 2014.

[LE14] Jinn-Liang Liu and Bob Eisenberg. Poisson-nernst-planck-fermi theory for modeling biological ion channels. *The Journal of chemical physics*, 141(22):12B640_1, 2014.

[LE15] Jinn-Liang Liu and Bob Eisenberg. Numerical methods for a poisson-nernst-planck-fermi model of biological ion channels. *Physical Review E*, 92(1):012711, 2015.

[Les14] Dave Lester. stdfix-exp.c: An implementation of exponential for fixpoint accum, 2014.

[LGQO18] Weiqiang Liu, Chongyan Gu, Gang Qu, and Máire ONeill. Approximate computing and its application to hardware security. In *Cyber-Physical Systems Security*, pages 43–67. Springer, 2018.

[LH10] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.

[LJL+18] Andrew Lines, Prasad Joshi, Ruokun Liu, Steve McCoy, Jonathan Tse, Yi-Hsin Weng, and Mike Davies. Loihi asynchronous neuromorphic research chip. In *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 32–33. IEEE, 2018.

[Lli88] Rodolfo R Llinás. The intrinsic electrophysiological properties of mammalian neurons: insights into central nervous system function. *Science*, 242(4886):1654–1664, 1988.

[Lli14] Rodolfo R Llinás. Intrinsic electrical properties of mammalian neurons and cns function: a historical perspective. *Frontiers in cellular neuroscience*, 8:320, 2014.

[LLM+18] Charl Linssen, Mikkel Elle Lepperd, Jessica Mitchell, Jari Pronold, Jochen Martin Eppler, Chrisitan Keup, Alexander Peyser, Susanne Kunkel, Philipp Weidel, Yannick Nodem, Dennis Terhorst, Rajalekshmi Deepu, Moritz Deger, Jan Hahne, Ankur Sinha, Alberto Antonietti, Maximilian Schmidt, Luciano Paz, Jess Garrido, Tammo Ippen, Luis Riquelme, Alex Serenko, Tobias Khn, Itaru Kitayama, Hkon Mrk, Sebastian Spreizer, Jakob Jordan, Jeyashree Krishnan, Mario Senden, Espen Hagen, Alexey Shusharin, Stine Brekke Vennemo, Dimitri Rodarie, Abigail Morrison, Steffen Graber, Jannis Schuecker, Sandra Diaz, Barna Zajzon, and Hans Ekkehard Plesser. Nest 2.16.0, August 2018.

[LLW+04] Cheng-yu Li, Jiang-teng Lu, Chien-ping Wu, Shu-min Duan, and Muming Poo. Bidirectional modification of presynaptic neuronal excitability accompanying spike timing-dependent synaptic plasticity. *Neuron*, 41(2):257–268, 2004.

[LSH+02] Michael London, Adi Schreibman, Michael Häusser, Matthew E Larkum, and Idan Segev. The information efficacy of a synapse. *Nature neuroscience*, 5(4):332, 2002.

[LWC+18a] C. Lin, A. Wild, G. N. Chinya, Y. Cao, M. Davies, D. M. Lavery, and H. Wang. Programming spiking neural networks on intels loihi. *Computer*, 51(3):52–61, March 2018.

[LWC+18b] Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Tsung-Han Lin, Mike Davies, and Hong Wang. Mapping spiking neural networks onto a manycore neuromorphic architecture. In *ACM SIGPLAN Notices*, volume 53, pages 78–89. ACM, 2018.

[Maa95] Wolfgang Maass. On the computational power of noisy spiking neurons, 1995.

[Maa96] Wolfgang Maass. Lower bounds for the computational power of networks of spiking neurons. *Neural computation*, 8(1):1–40, 1996.

[Maa97] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997.

[MAAI+14] 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, 2014.

[Mar11] Akira Maruoka. *Concise guide to computation theory*. Springer Science & Business Media, 2011.

[Mar18] Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

[May18] M. Mayberry. Intel creates neuromorphic research community to advance loihi test chip, 1 March 2018.

[MB97] David A McCormick and Thierry Bal. Sleep and arousal: thalamocortical mechanisms. *Annual review of neuroscience*, 20(1):185–215, 1997.

[MB00] R. Malaka and S. Buck. Solving nonlinear optimization problems using networks of spiking neurons. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 6, pages 486–491 vol.6, 2000.

[MB09] Julien Modolo and Anne Beuter. Linking brain dynamics, neural mechanisms, and deep brain stimulation in parkinson's disease: An integrated perspective. *Medical Engineering And Physics*, 31(6):615–623, 2009.

[MD91] Misha Mahowald and Rodney Douglas. A silicon neuron. *Nature*, 354(6354):515, 1991.

[Mea90] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[MHB08] J. Modolo, J. Henry, and A. Beuter. Dynamics of the subthalamo-pallidal complex in parkinson's disease during deep brain stimulation. *Journal of Biological Physics*, 34(3):251–266, Aug 2008.

[MHF19] Christian Mayr, Sebastian Hoeppner, and Steve Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning. *arXiv preprint arXiv:1911.02385*, 2019.

[MHSM12] Qingyun Ma, Mohammad Rafiqul Haider, Vinaya Lal Shrestha, and Yehia Massoud. Bursting hodgkin–huxley model-based ultra-low-power neuromimetic silicon neuron. *Analog Integrated Circuits and Signal Processing*, 73(1):329–337, Oct 2012.

[Mik20] Mantas Mikaitis. Issues with rounding in the gcc implementation of the iso 18037: 2008 standard fixed-point arithmetic. *arXiv preprint arXiv:2001.01496*, 2020.

[MLS+18] Mantas Mikaitis, David R Lester, Delong Shang, Steve Furber, Gengting Liu, Jim Garside, Stefan Scholze, Sebastian Höppner, and Andreas Dixius. Approximate fixed-point elementary function accelerator for the spinnaker-2 neuromorphic chip. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 37–44. IEEE, 2018.

[MMI13] Hesham Mostafa, Lorenz K Müller, and Giacomo Indiveri. Recurrent networks of coupled winner-take-all oscillators for solving constraint satisfaction problems. In *Advances in neural information processing systems*, pages 719–727, 2013.

[MMI15a] Hesham Mostafa, Lorenz K Müller, and Giacomo Indiveri. An event-based architecture for solving constraint satisfaction problems. *Nature communications*, 6:8941, 2015.

[MMI15b] Hesham Mostafa, Lorenz K Müller, and Giacomo Indiveri. Rhythmic inhibition allows neural networks to search for maximally consistent states. *Neural computation*, 2015.

[Mou57] Vernon B. Mountcastle. Modality and topographic properties of single neurons of cat's somatic sensory cortex. *Journal of Neurophysiology*, 20(4):408–434, 1957.

[MP11] M. Mahvash and A. C. Parker. Modeling intrinsic ion-channel and synaptic variability in a cortical neuromorphic circuit. In *2011 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 69–72, Nov 2011.

[MPGKF18] Mantas Mikaitis, Garibaldi Pineda García, James C Knight, and Steve B Furber. Neuromodulated synaptic plasticity on the spinnaker neuromorphic system. *Frontiers in neuroscience*, 12:105, 2018.

[MQSI18] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems*, 12(1):106–122, Feb 2018.

[MV17] Matthias Möller and Cornelis Vuik. On the impact of quantum computing technology on future developments in high-performance scientific computing. *Ethics and Information Technology*, 19(4):253–269, 2017.

[Nor09] Peter Norvig. Solving every sudoku puzzle. *Preprint*, 2009.

[NVS16] Robert A Nawrocki, Richard M Voyles, and Sean E Shaheen. A mini review of neuromorphic architectures and implementations. *IEEE Transactions on Electron Devices*, 63(10):3819–3829, 2016.

[OML19] Roman Orus, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: overview and prospects. *Reviews in Physics*, page 100028, 2019.

[Pep17] Ferdinand Peper. The end of moores law: Opportunities for natural computing? *New Generation Computing*, 35(3):253–269, 2017.

[PHE+17] Johannes Partzsch, Sebastian Höppner, Matthias Eberlein, Rene Schüffny, Christian Mayr, David R Lester, and Steve Furber. A fixed point exponential function accelerator for a neuromorphic many-core system. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[PIR+19] Christoph Pokorny, Matias J Ison, Arjun Rao, Robert Legenstein, Christos Papadimitriou, and Wolfgang Maass. STDP forms associations between memory traces in networks of spiking neurons. *bioRxiv*, page 188938, 2019.

[PK87] Ronald Pethig and Douglas B Kell. The passive electrical properties of biological systems: their significance in physiology, biophysics and biotechnology. *Physics in Medicine & Biology*, 32(8):933, 1987.

[PM98] Christophe Pouzat and Alain Marty. Autaptic inhibitory currents recorded from interneurones in rat cerebellar slices. *The Journal of Physiology*, 509(3):777–783, 1998.

[PM99] Christophe Pouzat and Alain Marty. Somatic recording of gabaergic autoreceptor current in cerebellar stellate and basket cells. *Journal of Neuroscience*, 19(5):1675–1690, 1999.

[PPG+13] Eustace Painkras, Luis A. Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R. Lester, Andrew D. Brown, and Steve B. Furber. SpiNNaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE J. Solid-State Circuits*, 48(8):1943–1953, aug 2013.

[PRM19] Marcel A. Pinto, Osvaldo A. Rosso, and Fernanda S. Matias. Inhibitory autapse mediates anticipated synchronization between coupled neurons. *Phys. Rev. E*, 99:062411, Jun 2019.

[Qui16] Rodrigo Quian Quiroga. Neuronal codes for visual perception and memory. *Neuropsychologia*, 83:227–241, 2016.

[RABI04] Benoit Roux, Toby Allen, Simon Berneche, and Wonpil Im. Theoretical and computational models of biological ion channels. *Quarterly reviews of biophysics*, 37(1):15–103, 2004.

[RBB+18] Oliver Rhodes, Petruţ A Bogdan, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R Lester, Mantas Mikaitis, Luis A Plana, Andrew GD Rowley, et al. spynnaker: A software package for running pynn simulations on spinnaker. *Frontiers in neuroscience*, 12, 2018.

[RBD+18] Andrew GD Rowley, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R Lester, Luis A Plana, Oliver Rhodes, Alan B Stokes, and Steve B Furber. Spinntools: The execution engine for the spinnaker platform. *arXiv preprint arXiv:1810.06835*, 2018.

[RCA+03] Michaël Russier, Edmond Carlier, Norbert Ankri, Laure Fronzaroli, and Dominique Debanne. A-, t-, and h-type currents shape intrinsic firing of developing rat abducens motoneurons. *The Journal of physiology*, 549(1):21–36, 2003.

[RD99] Stefan Rotter and Markus Diesmann. Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biological cybernetics*, 81(5-6):381–402, 1999.

[Rég04] Jean-Charles Régin. Global constraints and filtering algorithms. In *Constraint and Integer Programming*, pages 89–135. Springer, 2004.

[RG02] J Amiel Rosenkranz and Anthony A Grace. Dopamine-mediated modulation of odour-evoked amygdala potentials during pavlovian conditioning. *Nature*, 417(6886):282, 2002.

[RKG+11] Rajnish Ranjan, Georges Khazen, Luca Gambazzi, Srikanth Ramaswamy, Sean L Hill, Felix Schürmann, and Henry Markram. Channelpedia: an integrative and interactive database for ion channels. *Frontiers in neuroinformatics*, 5:36, 2011.

[RN16] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[RNS+17] Michael W. Reimann, Max Nolte, Martina Scolamiero, Katharine Turner, Rodrigo Perin, Giuseppe Chindemi, Pawe Dotko, Ran Levi, Kathryn Hess, and Henry Markram. Cliques of neurons bound into cavities provide a missing link between structure and function. *Frontiers in Computational Neuroscience*, 11:48, 2017.

[Rob16] S Ian Robertson. *Problem solving: perspectives from cognition and neuroscience*. Psychology Press, 2016.

[RPR+20] Oliver Rhodes, Luca Peres, Andrew Rowley, Andrew Gait, Luis A. Plana, Christian Brenninkmeijer, and Steve Furber. Real-time cortical simulation on neuromorphic hardware. *Royal Society of London. Proceedings A. Mathematical, Physical and Engineering Sciences*, 378(2164):1–21, 2020.

[RSD18] Ueli Rutishauser, Jean-Jacques Slotine, and Rodney J Douglas. Solving constraint-satisfaction problems with distributed neocortical-like neuronal networks. *Neural computation*, 30(5):1359–1393, 2018.

[SAKY01] Hailing Su, Gil Alroy, Eilon D Kirson, and Yoel Yaari. Extracellular calcium modulates persistent sodium current-dependent burst-firing in hippocampal pyramidal neurons. *Journal of Neuroscience*, 21(12):4173–4182, 2001.

[Sar18] Vasanth Sarathy. Real world problem-solving. *Frontiers in human neuroscience*, 12, 2018.

[SBG+10] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1947–1950, May 2010.

[SBG+15] Matthias Stadler, Nicolas Becker, Markus Gödker, Detlev Leutner, and Samuel Greiff. Complex problem solving and intelligence: A meta-analysis. *Intelligence*, 53:92–101, 2015.

[Sch15] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[SF02] Jens Spars and Steve Furber. *Principles asynchronous circuit design*. Springer, 2002.

[SGGW11] David Sterratt, Bruce Graham, Andrew Gillies, and David Willshaw. *Principles of computational modelling in neuroscience*. Cambridge University Press, 2011.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[Sig14] Daniel Sigg. Modeling ion channels: Past, present, and future. *The Journal of general physiology*, 144(1):7–26, 2014.

[SN84] Bert Sakmann and Erwin Neher. Patch clamp techniques for studying ionic channels in excitable membranes. *Annual review of physiology*, 46(1):455–472, 1984.

[SPP+17] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017.

[SQB99] Daniel Sigg, Hong Qian, and Francisco Bezanilla. Kramers diffusion theory applied to gating kinetics of voltage-dependent ion channels. *Biophysical Journal*, 76(2):782–803, 1999.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[Ste67] Richard B Stein. Some models of neuronal variability. *Biophysical journal*, 7(1):37–68, 1967.

[TAM94] Gina Turrigiano, LF Abbott, and Eve Marder. Activity-dependent changes in the intrinsic properties of cultured neurons. *Science*, 264(5161):974–977, 1994.

[TBMK16] Giulio Tononi, Melanie Boly, Marcello Massimini, and Christof Koch. Integrated information theory: from consciousness to its physical substrate. *Nature Reviews Neuroscience*, 17(7):450, 2016.

[TBS97] Gábor Tamás, Eberhard H. Buhl, and Peter Somogyi. Massive autaptic self-innervation of gabaergic neurons in cat visual cortex. *Journal of Neuroscience*, 17(16):6352–6364, 1997.

[TC11] Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.

[Tea20a] SpiNNaker Software Development Team. Software for spinnaker. `http://spinnakermanchester.github.io/`, accessed January 30, 2020.

[Tea20b] SpiNNaker Software Development Team. Spinnakermanchester/spynnaker. `https://github.com/SpiNNakerManchester/sPyNNaker/blob/master/neural_modelling/src/neuron/synapse_types/synapse_types_alpha_impl.h`, accessed January 30, 2020.

[Tea20c] SpiNNaker Software Development Team. SpiNNakerManchester/sPyNNaker. `https://github.com/SpiNNakerManchester/sPyNNaker/blob/master/neural_modelling/src/neuron/synapse_types/synapse_types_dual_excitatory_exponential_impl.h`, accessed January 30, 2020.

[TGK96] Donald G Truhlar, Bruce C Garrett, and Stephen J Klippenstein. Current status of transition-state theory. *The Journal of physical chemistry*, 100(31):12771–12800, 1996.

[TLD17] Ping Tak Peter Tang, Tsung-Han Lin, and Mike Davies. Sparse coding by spiking neural networks: Convergence theory and computational results. *arXiv preprint arXiv:1705.05475*, 2017.

[Tom15] Adam R Tomkins. *Action selection in the striatum: Implications for Huntington's disease*. PhD thesis, University of Sheffield, 2015.

[TSM19] Guangzhi Tang, Arpit Shah, and Konstantinos P Michmizos. Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. *arXiv preprint arXiv:1903.02504*, 2019.

[TT14] M Emin Tagluk and Ramazan Tekin. The influence of ion concentrations on the dynamic behavior of the hodgkin–huxley model-based cortical network. *Cognitive neurodynamics*, 8(4):287–298, 2014.

[TTBP91] Susan C Tucker, Mark E Tuckerman, Bruce J Berne, and Eli Pollak. Comparison of rate theories for generalized langevin dynamics. *The Journal of chemical physics*, 95(8):5809–5826, 1991.

[vARS+18] Sacha J. van Albada, Andrew G. Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B. Stokes, David R. Lester, Markus Diesmann, and Steve B. Furber. Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model. *Frontiers in Neuroscience*, 12:291, 2018.

[VB91] CA Vandenberg and F Bezanilla. A sodium channel gating model based on single channel, macroscopic ionic, and gating currents in the squid giant axon. *Biophysical journal*, 60(6):1511–1533, 1991.

[VCR03] Mariana Vargas-Caballero and Hugh PC Robinson. A slow fraction of mg2+ unblock of nmda receptors limits their contribution to spike generation in cortical pyramidal neurons. *Journal of neurophysiology*, 89(5):2778–2783, 2003.

[VDLG72] Hendrik Van Der Loos and Edmund M Glaser. Autapses in neocortex cerebri: synapses between a pyramidal cell's axon and its own dendrites. *Brain research*, 48:355–360, 1972.

[vWvHW04] Ingrid van Welie, Johannes A van Hooft, and Wytse J Wadman. Homeostatic scaling of neuronal excitability by synaptic modulation of somatic hyperpolarization-activated ih channels. *Proceedings of the national academy of sciences*, 101(14):5123–5128, 2004.

[Wal00] Toby Walsh. Sat v csp. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, CP '02, pages 441–456, London, UK, UK, 2000. Springer-Verlag.

[WBK10a] Simei Gomes Wysoski, Lubica Benuskova, and Nikola Kasabov. Evolving spiking neural networks for audiovisual information processing. *Neural Networks*, 23(7):819 – 835, 2010.

[WBK10b] Simei Gomes Wysoski, Lubica Benuskova, and Nikola Kasabov. Evolving spiking neural networks for audiovisual information processing. *Neural Networks*, 23(7):819 – 835, 2010.

[WKM+19] Timo Wunderlich, Akos Ferenc Kungl, Eric Müller, Andreas Hartel, Yannik Stradmann, Syed Ahmed Aamir, Andreas Grübl, Arthur Heimbrecht,

Korbinian Schreiber, David Stöckel, et al. Demonstrating advantages of neuromorphic computation: a pilot study. *Frontiers in neuroscience*, 13:260, 2019.

[WL03] X Wang and Nevin A Lambert. Membrane properties of identified lateral and medial perforant pathway projection neurons. *Neuroscience*, 117(2):485–492, 2003.

[WR17] Inc. Wolfram Research. Mathematica, Version 11.1, 2017. Champaign, IL, 2017.

[YJG03] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[ZL03] Wei Zhang and David J Linden. The other side of the engram: experience-driven changes in neuronal intrinsic excitability. *Nature Reviews Neuroscience*, 4(11):885, 2003.

[ZR02] Robert S Zucker and Wade G Regehr. Short-term synaptic plasticity. *Annual review of physiology*, 64(1):355–405, 2002.

Appendix A

Constraint Satisfaction In Loihi: Listings

Listing A.1: Python SNIP for on-chip validation.

```
setup_snips(board):
    """Setup snips to notify solution, stop simulation and read network state. """
    # Define SNIP process which sends information from lmt counter register to
    superhost about summation spikes.
    MANAGEMENT_PROCESS = create process(name="runMgmt",
                                        C file=path to  C_FILE,
                                        Management function="run_mgmt",
                                        Guard function="do_run_mgmt",
                                        phase=management,
                                        Chip ID=0,
                                        LMT ID=0)
    Call create_notification_channels(board)

create_notification_channels(board):
    """Create channels for communication between LMT and superhost
    notification_channel ← create channel from board
    # connect channel from LMT to superhost to send spike count from LMT register
    notification_channel.connect(MANAGEMENT_PROCESS, Host)

    # Create a channel for acknowledging superhost reception of spike time
    acknowledgement_channel ← create a channel from board
    # create notification_channel from LMT to superhost which is receiving
    spike count from LMT register
    acknowledgement_channel.connect(Host, MANAGEMENT_PROCESS)
```

Listing A.2: Python SNIP for on-chip validation.

```
def _connect_sumation_neuron_to_lmt(self):
    """Setup axon from summation neuron to LMT but do not read from host."""
    prob_condition <- time to start=runtime*2
    if number of summation neurons >1:
```

```
        summation_probes = SUMMATION_NEURON.probe(spikes, prob_condition)
    else:
        summation_probes = SUMMATION_NEURONS[-1].probe(spikes, prob_condition)
    return summation_probes

def chose_and_configure_readout_mechanism(mechanism="autapse"):
    if mechanism=="traces":
        enable activity traces (Impulse, decay)
    else if mechanism=="autapse"
        create autapse from 𝒯 to ℳ (weight, box duration)
```

Listing A.3: C Guard SNIP for on-chip validation.

```
int SPIKE_PROBE_ID = probe ID in LMT

// Run SNIP from the start of simulation and get channels IDs.
int do_run_management(simulation state){
        if timestep > 0 :
            int CHANNEL_ID_NOTIFY  ← get ID for notification channel from LMT to Host
            int CHANNEL_ID_ACKNOWLEDGE ← get ID for acknowledgeement channel from Host to LMT
        if timestep > 2:
            return 1
        else:
            int SPIKE_COUNT  ← get spike counter from simulation state
            Reset SPIKE_COUNT to 0
            return 0
        }
```

Listing A.4: Main C SNIP for on-chip validation.

```
// Read spike counter, reset counter to zero and write count to channel
    void run_management(simulation state) {
         SPIKE_COUNT ← get spike counter from simulation state

         if SPIKE_COUNT==1:
            SPIKE_COUNT = 0;    // Avoid overflow
            int solution time= current timestep;
            write solution time to CHANNEL_ID_NOTIFY
            listen/wait CHANNEL_ID_ACKNOWLEDGE for acknowledgement from LMT

         if time_step == total_steps:
            write -1 to CHANNEL_ID_NOTIFY // Write -1 to channel if no solution time registered
            listen/wait CHANNEL_ID_ACKNOWLEDGE for acknowledgement from LMT
            printf(End of simulation was reached)
        }
```

Appendix B

Details on Linearisability of Postsynaptic Currents

B.1 Details on Alpha PSC

B.1.1 An Example with Two Spikes

Let us first consider two spikes arriving at times t_1 and t_2 at the postsynaptic neuron α, producing alpha postsynaptic currents:

$$j_1(t) = q\frac{t-t_1}{\tau^2}\left(e^{-\frac{t-t_1}{\tau}}\right)\Theta(t-t_1), \tag{B.1}$$

$$j_2(t) = q\frac{t-t_2}{\tau^2}\left(e^{-\frac{t-t_2}{\tau}}\right)\Theta(t-t_2). \tag{B.2}$$

Denoting $\Theta(t-t_i)$ by Θ_i and inserting equations B.1, B.2 in 5.1 and assuming $\omega_{\alpha\beta} = 1$, we have:

$$
\begin{aligned}
J(t) &= j_1(t) + j_2(t) \\
&= q\frac{t-t_1}{\tau^2}\Theta_1\left(\mathrm{e}^{-\frac{t-t_1}{\tau}}\right) + q\frac{t-t_2}{\tau^2}\Theta_2\left(\mathrm{e}^{-\frac{t-t_2}{\tau}}\right) \\
&= q\left[\frac{t}{\tau^2} - \frac{t_1}{\tau^2}\right]\Theta_1\left(\mathrm{e}^{-\frac{t}{\tau}}\mathrm{e}^{\frac{t_1}{\tau}}\right) + q\left[\frac{t}{\tau^2} - \frac{t_2}{\tau^2}\right]\Theta_2\left(\mathrm{e}^{-\frac{t}{\tau}}\mathrm{e}^{\frac{t_2}{\tau}}\right) \\
&= \left(\left[t-t_1\right]\Theta_1\mathrm{e}^{\frac{t_1}{\tau}} + \left[t-t_2\right]\Theta_2\mathrm{e}^{\frac{t_2}{\tau}}\right)\frac{q}{\tau^2}\mathrm{e}^{-\frac{t}{\tau}} \\
&= \left(t\Theta_1\mathrm{e}^{\frac{t_1}{\tau}} - t_1\Theta_1\mathrm{e}^{\frac{t_1}{\tau}} + t\Theta_2\mathrm{e}^{\frac{t_2}{\tau}} - t_2\Theta_2\mathrm{e}^{\frac{t_2}{\tau}}\right)\frac{q}{\tau^2}\mathrm{e}^{-\frac{t}{\tau}} \\
\Rightarrow J(t) &= \left(\left[\Theta_1\mathrm{e}^{\frac{t_1}{\tau}} + \Theta_2\mathrm{e}^{\frac{t_2}{\tau}}\right]t - \left[t_1\Theta_1\mathrm{e}^{\frac{t_1}{\tau}} - t_2\Theta_2\mathrm{e}^{\frac{t_2}{\tau}}\right]\right)\frac{q}{\tau^2}\mathrm{e}^{-\frac{t}{\tau}}, \qquad \text{(B.3)} \\
&= (At - B)\frac{q}{\tau^2}\mathrm{e}^{-\frac{t}{\tau}} \qquad \text{(B.4)}
\end{aligned}
$$

were we have recognized the terms inside square brackets in the penultimate step as a set of piece-wise constants (one value for each temporal domain) defined by:

$$A = \left[\Theta_1\mathrm{e}^{\frac{t_1}{\tau}} + \Theta_2\mathrm{e}^{\frac{t_2}{\tau}}\right], \qquad \text{(B.5)}$$

$$B = \left[t_1\Theta_1\mathrm{e}^{\frac{t_1}{\tau}} - t_2\Theta_2\mathrm{e}^{\frac{t_2}{\tau}}\right]. \qquad \text{(B.6)}$$

Thus:

$$
\begin{aligned}
J(t > t_1) &= \left(t - \frac{B}{A}\right)\frac{q}{\tau^2}A\mathrm{e}^{-\frac{t}{\tau}} \\
&= \frac{q}{\tau^2}\left(t - \frac{B}{A}\right)\mathrm{e}^{\frac{\tau}{\tau}ln(A)}\mathrm{e}^{-\frac{t}{\tau}} \\
&= \frac{q\left(t - \frac{B}{A}\right)}{\tau^2}\mathrm{e}^{-\frac{t-\tau ln(A)}{\tau}}. \qquad \text{(B.7)}
\end{aligned}
$$

We have restricted $t > t_1$ because before any spike arrives the fraction $\frac{B}{A}$ is undetermined, but from equation B.4 we do know that $J(t < t_1) = 0$. In B.7 the terms $\frac{B}{A}$ and $\tau ln(A)$ remain constant (piece-wise), so we have arrived to the same functional form as the one of equation 5.4. Notice that A and B contain the Heaviside functions Θ_1 and Θ_2,

this imply the definition of three time domains for J(t):

$$\begin{array}{rlllll} t < t_1, & \text{where} & \Theta_1 = 0 & \text{and} & \Theta_2 & = 0, \\ t_1 < t < t_2, & \text{where} & \Theta_1 = 1 & \text{and} & \Theta_2 & = 0, \\ \text{and} \quad t > t_2, & \text{where} & \Theta_1 = 1 & \text{and} & \Theta_2 & = 1. \end{array} \tag{B.8}$$

Let us see what the form of A and B for each region:

$$\begin{array}{rlll} t < t_1: & A = 0 & \text{and } B = 0, \\ t_1 < t < t_2: & A = \mathrm{e}^{\frac{t_1}{\tau}}, & \text{and } B = t_1 \mathrm{e}^{\frac{t_1}{\tau}}, \\ t > t_2: & A = \mathrm{e}^{\frac{t_1}{\tau}} + \mathrm{e}^{\frac{t_2}{\tau}} & \text{and } B = t_1 \mathrm{e}^{\frac{t_1}{\tau}} - t_2 \mathrm{e}^{\frac{t_2}{\tau}}. \end{array} \tag{B.9}$$

Thus, substitution of these set of conditions in B.7 defines $J(t > t_1)$ for each region (note that $\frac{B}{A} = t_1$ for $t_1 < t < t_2$.):

$$t < t_1: \qquad J(t) = 0, \tag{B.10}$$

$$t_1 < t < t_2: \qquad J(t) = \frac{q\,(t - t_1)}{\tau^2} \mathrm{e}^{-\frac{t-t_1}{\tau}}, , \tag{B.11}$$

$$t > t_2: \qquad J(t) = \frac{q\left(t - \frac{[t_1 \mathrm{e}^{\frac{t_1}{\tau}} - t_2 \mathrm{e}^{\frac{t_2}{\tau}}]}{[\mathrm{e}^{\frac{t_1}{\tau}} + \mathrm{e}^{\frac{t_2}{\tau}}]}\right)}{\tau^2} \mathrm{e}^{-\frac{t - \tau ln(\mathrm{e}^{\frac{t_1}{\tau}} + \mathrm{e}^{\frac{t_2}{\tau}})}{\tau}}. \tag{B.12}$$

before any spike arrives $J = 0$, when just the first spike arrives at t_1 equation B.11 reduces to equation B.1. Finally, equation B.12 gives the postsynaptic current when both spikes are present after time t_2 (see figure 5.2a). To better write equation B.12, let us define $t_\gamma = \tau ln([\mathrm{e}^{\frac{t_1}{\tau}} + \mathrm{e}^{\frac{t_2}{\tau}}])$ and $t_\kappa = \frac{[t_1 \mathrm{e}^{\frac{t_1}{\tau}} - t_2 \mathrm{e}^{\frac{t_2}{\tau}}]}{[\mathrm{e}^{\frac{t_1}{\tau}} + \mathrm{e}^{\frac{t_2}{\tau}}]}$, now:

$$J(t) = \frac{q\,(t - t_\kappa)}{\tau^2} \mathrm{e}^{-\frac{t - t_\gamma}{\tau}}. \tag{B.13}$$

B.1.2 Generalization to N Spikes

The procedure of B.1.1 can be generalized for an arbitrary number of spikes N at times t_i, each of them producing a postsynaptic current $j_i(t)$. From equations 5.1 and 5.4, the total current on α for an arbitrary time t, is given by:

$$
\begin{aligned}
J(t) &= \sum_i j_i(t) \\
&= \sum_i q \frac{t-t_i}{\tau^2} \left(\mathrm{e}^{-\frac{t-t_i}{\tau}} \right) \Theta_{t_i} \\
&= \sum_i q \left[\frac{t}{\tau^2} - \frac{t_i}{\tau^2} \right] \mathrm{e}^{-\frac{t}{\tau}} \mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i} \\
&= \frac{q\mathrm{e}^{-\frac{t}{\tau}}}{\tau^2} \sum_i \left[t - t_i \right] \mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i} \\
&= \frac{q\mathrm{e}^{-\frac{t}{\tau}}}{\tau^2} \sum_i \left[t\mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i} - t_i \mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i} \right] \\
&= \frac{q\mathrm{e}^{-\frac{t}{\tau}}}{\tau^2} \left[t \sum_i \mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i} - \sum_i t_i \mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i} \right] \qquad \text{(B.14)} \\
&= \frac{q\mathrm{e}^{-\frac{t}{\tau}}}{\tau^2} \left(t\alpha - \beta \right) \qquad \text{(B.15)}
\end{aligned}
$$

Again we have recognized the summations as piece-wise constants for each temporal domain, for readability we have denoted them by $\alpha = \sum_i \mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i}$ and $\beta = \sum_i t_i \mathrm{e}^{\frac{t_i}{\tau}} \Theta_{t_i}$. We have now for any time after the first spike at t_0:

$$
\begin{aligned}
J(t > t_0) &= q \frac{\left(t - \frac{\beta}{\alpha} \right)}{\tau^2} \alpha \mathrm{e}^{-\frac{t}{\tau}} \\
&= q \frac{\left(t - \frac{\beta}{\alpha} \right)}{\tau^2} \mathrm{e}^{-\frac{t}{\tau}} \mathrm{e}^{ln(\alpha)} \\
&= q \frac{\left(t - \frac{\beta}{\alpha} \right)}{\tau^2} \mathrm{e}^{-\frac{t-\tau ln(\alpha)}{\tau}} \\
&= q \frac{\left(t - t_\gamma \right)}{\tau^2} \mathrm{e}^{-\frac{t-t_\kappa}{\tau}} \qquad \text{(B.16)}
\end{aligned}
$$

where we have written $t_\gamma = \frac{\beta}{\alpha}$ and $t_\kappa = \tau ln(\alpha)$, as they naturally appear as generalized versions of t_i, in fact, they seem to emerge from the dual role of t_i on the exponential and linear parts of equation 5.5

the open version of equation B.16 is:

$$J(t > t_0) = q\frac{\left(t - \frac{\sum_i t_i e^{\frac{t_i}{\tau}}\Theta_{t_i}}{\sum_i e^{\frac{t_i}{\tau}}\Theta_{t_i}}\right)}{\tau^2} e^{-\frac{t-\tau ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_{t_i})}{\tau}} \tag{B.17}$$

here, one can see how t is segmented in N+1 sections, defined by the particular values of the Heaviside functions. For an arbitrary region $t_{j-1} < t_j < t_{j+1}$, defined by the interval between spike j and spike j+1, the total current is thus expressed as:

$$J(t_j) = q\frac{\left(t - \frac{\sum_{i=0}^{j} t_i e^{\frac{t_i}{\tau}}}{\sum_{i=0}^{j} e^{\frac{t_i}{\tau}}}\right)}{\tau^2} e^{-\frac{t-\tau ln(\sum_{i=0}^{j} e^{\frac{t_i}{\tau}})}{\tau}} \tag{B.18}$$

Which can be written as equation B.13 with generalized times defined by:

$$t_\kappa = \frac{\sum_{i=0}^{j} t_i e^{\frac{t_i}{\tau}}}{\sum_{i=0}^{j} e^{\frac{t_i}{\tau}}}, \tag{B.19}$$

$$t_\gamma = \tau ln(\sum_{i=0}^{j} e^{\frac{t_i}{\tau}}). \tag{B.20}$$

B.1.3 Discrete Buffered Version

Here we derive the recurrence relations for updating the buffer values corresponding to the exponential and linear components of the generalized alpha function of equation 5.9. We begin with the exponential kernel as it will be needed for the linear updating at the spike times. We use the subindex n to account for the values at the current time t_n and $n+1$ for their values at the next time $t_n + \Delta_n$.

B.1.3.1 Exponential Buffer

Updating Between Spike

In order to obtain the relation for ε_{n+1} let us compute $\Delta\varepsilon$ from the definition of equation 5.5:

$$\begin{aligned}
\Delta\varepsilon &= \varepsilon_{n+1} - \varepsilon_n \\
&= e^{-\frac{t_{n+1}-\tau\ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_i)}{\tau}} - e^{-\frac{t_n-\tau\ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_i)}{\tau}} \\
&= e^{-\frac{t_{n+1}}{\tau}} e^{\frac{\tau\ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_i)}{\tau}} - e^{-\frac{t_n}{\tau}} e^{\frac{\tau\ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_i)}{\tau}} \\
&= e^{-\frac{t_{n+1}}{\tau}} \sum_i e^{\frac{t_i}{\tau}}\Theta_i - e^{-\frac{t_n}{\tau}} \sum_i e^{\frac{t_i}{\tau}}\Theta_i \\
&= \sum_i e^{\frac{t_i}{\tau}}\Theta_i \left[e^{-\frac{t_{n+1}}{\tau}} - e^{-\frac{t_n}{\tau}}\right] \\
&= \sum_i e^{\frac{t_i}{\tau}}\Theta_i \left[e^{-\frac{t_n+\Delta t}{\tau}} - e^{-\frac{t_n}{\tau}}\right] \\
&= \sum_i e^{\frac{t_i}{\tau}}\Theta_i \left[e^{-\frac{\Delta t}{\tau}} - 1\right] e^{-\frac{t_n}{\tau}} \\
&= e^{-\frac{t_n}{\tau}} e^{\frac{\tau}{\tau}\ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_i)} \left[e^{-\frac{\Delta t}{\tau}} - 1\right] \\
&= e^{-\frac{t_n-\tau\ln(\sum_i e^{\frac{t_i}{\tau}}\Theta_i)}{\tau}} \left[e^{-\frac{\Delta t}{\tau}} - 1\right] \\
&= \varepsilon_n \left[e^{-\frac{\Delta t}{\tau}} - 1\right]
\end{aligned} \tag{B.21}$$

equation B.21 implies that the updating rule for each continuous segment of the exponential buffer is:

$$\varepsilon_{n+1} = \varepsilon_n e^{-\frac{\Delta t}{\tau}} \tag{B.22}$$

B.1.3.2 Updating at Spike Times

To get the expression for updating the buffer value at the time of a new spike arrival, let us say spike number η, we use again the definition 5.5, but keeping in mind that at t_n only $\eta - 1$ spikes have arrived whilst at t_{n+1} the η should be taken into account. We

have also that $t_{n+1} = t_\eta$ then:

$$\begin{aligned}
\varepsilon_{n+1} &= e^{-\frac{t_{n+1}-\tau\ln(\sum_i^\eta e^{\frac{t_i}{\tau}}\Theta_i)}{\tau}} \\
&= e^{-\frac{t_{n+1}}{\tau}}\sum_i^\eta e^{\frac{t_i}{\tau}}\Theta_i \\
&= e^{-\frac{\Delta t}{\tau}}e^{-\frac{t_n}{\tau}}\left[\sum_i^{\eta-1} e^{\frac{t_i}{\tau}} + e^{\frac{t_\eta}{\tau}}\right] \\
&= e^{-\frac{\Delta t}{\tau}}e^{-\frac{t_n}{\tau}}\sum_i^{\eta-1} e^{\frac{t_i}{\tau}} + e^{-\frac{t_n}{\tau}}e^{-\frac{\Delta t}{\tau}}e^{\frac{t_\eta}{\tau}} \\
&= e^{-\frac{\Delta t}{\tau}}e^{-\frac{t_n}{\tau}}e^{\frac{\tau}{\tau}\ln(\sum_i^{\eta-1} e^{\frac{t_i}{\tau}})} + e^{-\frac{t_{n+1}}{\tau}}e^{\frac{t_\eta}{\tau}} \\
&= e^{-\frac{\Delta t}{\tau}}e^{-\frac{t_n-\tau\ln(\sum_i^{\eta-1} e^{\frac{t_i}{\tau}}\Theta_i)}{\tau}} + 1
\end{aligned} \tag{B.23}$$

where we recognize the second exponential as ε_n, hence:

$$\varepsilon_{n+1} = e^{-\frac{\Delta t}{\tau}}\varepsilon_n + 1 \tag{B.24}$$

which is our recurrence relation under the presence of a new spike.

B.1.3.3 Linear Buffer

Updating Between Spike

From equation 5.10 we compute $\Delta\lambda = \lambda_{n+1} - \lambda_n$ as:

$$\begin{aligned}
\Delta\lambda &= \frac{q}{\tau^2}\left(t_{n+1} - \frac{\sum_i t_i e^{\frac{t_i}{\tau}}\Theta_i}{\sum_i e^{\frac{t_i}{\tau}}\Theta_i}\right) - \frac{q}{\tau^2}\left(t_n - \frac{\sum_i t_i e^{\frac{t_i}{\tau}}\Theta_i}{\sum_i e^{\frac{t_i}{\tau}}\Theta_i}\right) \\
&= \frac{q}{\tau^2}(t_{n+1} - t_n) \qquad \text{(B.25)} \\
&= \frac{q}{\tau^2}\Delta t \qquad \text{(B.26)}
\end{aligned}$$

thus the updating rule for each continuous segment of the linear kernel is given by:

$$\lambda_{n+1} = \lambda_n + \frac{q}{\tau^2}\Delta t \tag{B.27}$$

B.1.3.4 Updating at Spike Times

Similarly as we did with the exponential buffer, we proceed to calculate λ_{n+1} for the event of a new spike η arriving to the postsynaptic neuron α, using equation 5.10 we have:

$$
\begin{aligned}
\lambda_{n+1} &= \frac{q}{\tau^2}\left(t_{n+1} - \frac{\sum_i^N t_i e^{\frac{t_i}{\tau}}\Theta_i}{\sum_i^N e^{\frac{t_i}{\tau}}\Theta_i}\right) \\
\lambda_{n+1} &= \frac{q}{\tau^2}\left(t_{n+1} - \frac{\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}} + t_{n+1}e^{\frac{t_{n+1}}{\tau}}}{\sum_i^{N-1} e^{\frac{t_i}{\tau}} + e^{\frac{t_{n+1}}{\tau}}}\right) \\
&= \frac{q}{\tau^2}\left(t_n + \Delta t - \frac{\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}} + (t_n+\Delta t)e^{\frac{t_n+\Delta t}{\tau}}}{\sum_i^{N-1} e^{\frac{t_i}{\tau}} + e^{\frac{t_n+\Delta t}{\tau}}}\right) \\
&= \frac{q}{\tau^2}\left(t_n + \Delta t - \frac{\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}} + (t_n+\Delta t)e^{\frac{t_n+\Delta t}{\tau}}}{\left[e^{-\frac{t_n+\Delta t}{\tau}}\sum_i^{N-1} e^{\frac{t_i}{\tau}} + 1\right]e^{\frac{t_n+\Delta t}{\tau}}}\right)
\end{aligned}
\tag{B.28}
$$

where we recognize the term in square brackets as ε_{n+1} from equation B.24, thus:

$$\begin{aligned}
\lambda_{n+1} &= \frac{q}{\tau^2}\left(t_n+\Delta t-\frac{\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}+(t_n+\Delta t)e^{\frac{t_n+\Delta t}{\tau}}}{\varepsilon_{n+1}e^{\frac{t_n+\Delta t}{\tau}}}\right)\\
&= \frac{q}{\tau^2}\left(t_n+\Delta t-\frac{\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}}{\varepsilon_{n+1}e^{\frac{t_n+\Delta t}{\tau}}}-\frac{(t_n+\Delta t)e^{\frac{t_n+\Delta t}{\tau}}}{\varepsilon_{n+1}e^{\frac{t_n+\Delta t}{\tau}}}\right)\\
&= \frac{q}{\tau^2}\left(t_n+\Delta t-\frac{t_n}{\varepsilon_{n+1}}-\frac{\Delta t}{\varepsilon_{n+1}}-\frac{\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}}{\varepsilon_{n+1}e^{\frac{t_n+\Delta t}{\tau}}}\right)\\
&= \frac{q}{\tau^2}\left(\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right)+t_n\left(\frac{\varepsilon_{n+1}-1}{\varepsilon_{n+1}}\right)-\frac{e^{-\frac{t_n+\Delta t}{\tau}}\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}}{\varepsilon_{n+1}}\right)\\
&= \frac{q}{\tau^2}\left(\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right)+\frac{1}{\varepsilon_{n+1}}\left[t_n\varepsilon_{n+1}-t_n-e^{-\frac{t_n+\Delta t}{\tau}}\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}\right]\right)
\end{aligned} \tag{B.29}$$

let us recover the open expression of ε_{n+1} just for the term inside square brackets:

$$\begin{aligned}
\lambda n+1 &= \frac{q}{\tau^2}\left(\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right)+\frac{1}{\varepsilon_{n+1}}\left[t_n\left[e^{-\frac{t_n+\Delta t}{\tau}}\sum_i^{N-1} e^{\frac{t_i}{\tau}}+1\right]-t_n-e^{-\frac{t_n+\Delta t}{\tau}}\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}\right]\right)\\
&= \frac{q}{\tau^2}\left(\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right)+\frac{1}{\varepsilon_{n+1}}\left[t_n e^{-\frac{t_n+\Delta t}{\tau}}\sum_i^{N-1} e^{\frac{t_i}{\tau}}+t_n-t_n-e^{-\frac{t_n+\Delta t}{\tau}}\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}\right]\right)\\
&= \frac{q}{\tau^2}\left(\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right)+\frac{e^{-\frac{t_n+\Delta t}{\tau}}}{\varepsilon_{n+1}}\left[t_n\sum_i^{N-1} e^{\frac{t_i}{\tau}}-\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}\right]\right)
\end{aligned} \tag{B.30}$$

remembering that $(ac+b) = a(c+\frac{b}{a})$ we get a very convenient expression:

$$\lambda_{n+1} = \frac{q}{\tau^2}\left(\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right) + \frac{\left[-\frac{t_n+\Delta t}{\tau}\sum_i^{N-1} e^{\frac{t_i}{\tau}}\right]}{\varepsilon_{n+1}}\left[t_n - \frac{\sum_i^{N-1} t_i e^{\frac{t_i}{\tau}}}{\sum_i^{N-1} e^{\frac{t_i}{\tau}}}\right]\right) \tag{B.31}$$

where we recognize the first term in square brackets as $\varepsilon_{n+1}-1$ and the second as $\frac{\tau^2}{q}\lambda_n$, which allow us to make:

$$\begin{aligned}\lambda_{n+1} &= \frac{q}{\tau^2}\left(\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right) + \frac{\varepsilon_{n+1}-1}{\varepsilon_{n+1}}\lambda_n\frac{\tau^2}{q}\right)\\ &= \frac{q}{\tau^2}\Delta t\left(1-\frac{1}{\varepsilon_{n+1}}\right) + \left(1-\frac{1}{\varepsilon_{n+1}}\right)\lambda_n\\ &= \left[\frac{q}{\tau^2}\Delta t + \lambda_n\right]\left(1-\frac{1}{\varepsilon_{n+1}}\right).\end{aligned} \tag{B.32}$$

Equation B.32 is the updating rule for the linear buffer at the arrival of any new spike. Notice that it uses the updated value of the exponential buffer which is the reason for having deduced it first, remember from equation B.24 that $\varepsilon_{n+1} = e^{-\frac{\Delta t}{\tau}}\varepsilon_n + 1$ so it is known to us from the available information at t_n.

B.2 Details on Dual Exponential PSC

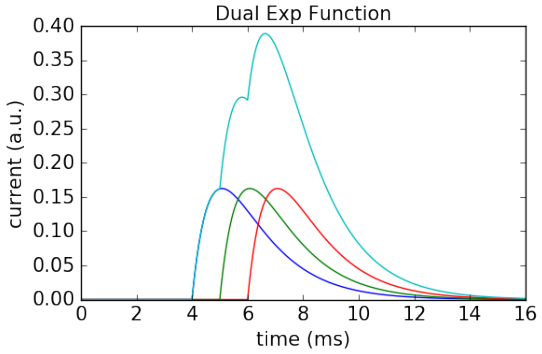


Figure B.1: Dual exponential postsynaptic currents generated by individual spikes arriving at times 4, 5 and 6 ms. The light blue line represents the total induced current.

Similarly, as we did in the alpha case, let us begin with two incoming spikes:

$$j_1(t) = \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-t_1}{\tau_f}} - \mathrm{e}^{-\frac{t-t_1}{\tau_r}}\right)\Theta(t-t_1) \tag{B.33}$$

$$j_2(t) = \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-t_2}{\tau_f}} - \mathrm{e}^{-\frac{t-t_2}{\tau_r}}\right)\Theta(t-t_2) \tag{B.34}$$

substituting B.34 in 5.1:

$$\begin{aligned}
J(t) &= j_1(t) + j_2(t) \\
&= \frac{q}{\tau_f - \tau_r}\left[\left(\mathrm{e}^{-\frac{t-t_1}{\tau_f}} - \mathrm{e}^{-\frac{t-t_1}{\tau_r}}\right)\Theta_1 + \left(\mathrm{e}^{-\frac{t-t_2}{\tau_f}} - \mathrm{e}^{-\frac{t-t_2}{\tau_r}}\right)\Theta_2\right] \\
&= \frac{q}{\tau_f - \tau_r}\left[\mathrm{e}^{-\frac{t-t_1}{\tau_f}}\Theta_1 - \mathrm{e}^{-\frac{t-t_1}{\tau_r}}\Theta_1 + \mathrm{e}^{-\frac{t-t_2}{\tau_f}}\Theta_2 - \mathrm{e}^{-\frac{t-t_2}{\tau_r}}\Theta_2\right] \\
&= \frac{q}{\tau_f - \tau_r}\left[\mathrm{e}^{-\frac{t}{\tau_f}}\mathrm{e}^{\frac{t_1}{\tau_f}}\Theta_1 - \mathrm{e}^{-\frac{t}{\tau_r}}\mathrm{e}^{\frac{t_1}{\tau_r}}\Theta_1 + \mathrm{e}^{-\frac{t}{\tau_f}}\mathrm{e}^{\frac{t_2}{\tau_f}}\Theta_2 - \mathrm{e}^{-\frac{t}{\tau_r}}\mathrm{e}^{\frac{t_2}{\tau_r}}\Theta_2\right] \\
&= \frac{q}{\tau_f - \tau_r}\left[\mathrm{e}^{-\frac{t}{\tau_f}}\left(\mathrm{e}^{\frac{t_1}{\tau_f}}\Theta_1 + \mathrm{e}^{\frac{t_2}{\tau_f}}\Theta_2\right) - \mathrm{e}^{-\frac{t}{\tau_r}}\left(\mathrm{e}^{\frac{t_1}{\tau_r}}\Theta_1 + \mathrm{e}^{\frac{t_2}{\tau_r}}\Theta_2\right)\right].
\end{aligned} \tag{B.35}$$

Recognizing the terms in parentheses as piece-wise constants, we can write: $A = \mathrm{e}^{\frac{t_1}{\tau_f}}\Theta_1 + \mathrm{e}^{\frac{t_2}{\tau_f}}\Theta_2$ and $B = \mathrm{e}^{\frac{t_1}{\tau_r}}\Theta_1 + \mathrm{e}^{\frac{t_2}{\tau_r}}\Theta_2$. Thus, equation B.35 becomes:

$$\begin{aligned}
J(t) &= \frac{q}{\tau_f - \tau_r}\left[\mathrm{e}^{-\frac{t}{\tau_f}}A - \mathrm{e}^{-\frac{t}{\tau_r}}B\right], \\
&= \frac{q}{\tau_f - \tau_r}\left[\mathrm{e}^{-\frac{t-\tau_f ln(A)}{\tau_f}} - \mathrm{e}^{-\frac{t-\tau_r ln(B)}{\tau_r}}\right].
\end{aligned} \tag{B.36}$$

As in the last section, let us analyze the time regions defined by the Heaviside functions inside A and B (e.g. by the spike times), which are the same as in equation B.8. This time A and B for each temporal domain are given by:

$$\begin{aligned}
t < t_1 : &\qquad A = 0 &&\text{and } B = 0, \\
t_1 < t < t_2 : &\qquad A = \mathrm{e}^{\frac{t_1}{\tau_f}}, &&\text{and } B = \mathrm{e}^{\frac{t_1}{\tau_r}}, \\
t > t_2 : &\qquad A = \mathrm{e}^{\frac{t_1}{\tau_f}} + \mathrm{e}^{\frac{t_2}{\tau_f}} &&\text{and } B = \mathrm{e}^{\frac{t_1}{\tau_r}} - \mathrm{e}^{\frac{t_2}{\tau_r}}.
\end{aligned} \tag{B.37}$$

The corresponding expressions for $J(t)$ are as follows:

$$t < t_1 : \qquad J(t) = 0, \tag{B.38}$$

$$t_1 < t < t_2 : \qquad J(t) = \frac{q}{\tau_f - \tau_r}\left[e^{-\frac{t-t_1}{\tau_f}} - e^{-\frac{t-t_1}{\tau_r}}\right], \tag{B.39}$$

$$t > t_2 : \qquad J(t) = \frac{q}{\tau_f - \tau_r}\left[e^{-\frac{t-\tau_f ln(e^{\frac{t_1}{\tau_f}} + e^{\frac{t_2}{\tau_f}})}{\tau_f}} - e^{-\frac{t-\tau_r ln(e^{\frac{t_1}{\tau_r}} - e^{\frac{t_2}{\tau_r}})}{\tau_r}}\right]. \tag{B.40}$$

Equation B.40 can be rewriting as:

$$J(t) = \frac{q}{\tau_f - \tau_r}\left[e^{-\frac{t-\zeta}{\tau_f}} - e^{-\frac{t-\lambda}{\tau_r}}\right]. \tag{B.41}$$

where ζ and λ are known constants.

B.2.1 Generalization to N Spikes

Generalization can be achieved for an arbitrary number of spikes N at times t_i, each of them producing a postsynaptic current $j_i(t)$. Using equations 5.1 and 5.3, we have for the the total current on α at an arbitrary time t:

$$\begin{aligned}
J(t) &= \sum_i j_i(t) \\
J(t) &= \sum_i \frac{q}{\tau_f - \tau_r}\left(e^{-\frac{t-t_i}{\tau_f}} - e^{-\frac{t-t_i}{\tau_r}}\right)\Theta_{t_i} \\
&= \frac{q}{\tau_f - \tau_r}\sum_i \left(e^{-\frac{t-t_i}{\tau_f}}\Theta_{t_i} - e^{-\frac{t-t_i}{\tau_r}}\Theta_{t_i}\right) \\
&= \frac{q}{\tau_f - \tau_r}\sum_i \left(e^{-\frac{t}{\tau_f}}e^{\frac{t_i}{\tau_f}}\Theta_{t_i} - e^{-\frac{t}{\tau_r}}e^{\frac{t_i}{\tau_r}}\Theta_{t_i}\right) \\
&= \frac{q}{\tau_f - \tau_r}\left(e^{-\frac{t}{\tau_f}}\sum_i e^{\frac{t_i}{\tau_f}}\Theta_{t_i} - e^{-\frac{t}{\tau_r}}\sum_i e^{\frac{t_i}{\tau_r}}\Theta_{t_i}\right),
\end{aligned} \tag{B.42}$$

rewriting the summations on the right hand as α and β:

$$\begin{aligned} J(t) &= \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t}{\tau_f}}\alpha - \mathrm{e}^{-\frac{t}{\tau_r}}\beta\right) \\ &= \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t}{\tau_f}}\mathrm{e}^{ln(\alpha)} - \mathrm{e}^{-\frac{t}{\tau_r}}\mathrm{e}^{ln(\beta)}\right) \\ &= \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-\tau_f ln(\alpha)}{\tau_f}} - \mathrm{e}^{-\frac{t-\tau_r ln(\beta)}{\tau_r}}\right) \\ &= \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-t_\gamma}{\tau_f}} - \mathrm{e}^{-\frac{t-t_\kappa}{\tau_r}}\right) \end{aligned} \tag{B.43}$$

where we have written $t_\gamma = \tau_f ln(\alpha)$ and $t_\kappa = \tau_r ln(\beta)$, arriving at the analogous of equation B.16, but for a dual exponential function.

In order to identify the time domains, let us open B.43 as:

$$J(t) = \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-\tau_f ln(\sum_i \mathrm{e}^{\frac{t_i}{\tau_f}}\Theta_{t_i})}{\tau_f}} - \mathrm{e}^{-\frac{t-\tau_r ln(\sum_i \mathrm{e}^{\frac{t_i}{\tau_r}}\Theta_{t_i})}{\tau_r}}\right). \tag{B.44}$$

Equation B.44 defines N+1 regions of time characterized by the respective values of Θ_{t_i} in each zone. This implies that for an arbitrary region $t_{j-1} < t_j < t_{j+1}$, equation B.44 takes the form:

$$J(t) = \frac{q}{\tau_f - \tau_r}\left(\mathrm{e}^{-\frac{t-\tau_f ln(\sum_{i=0}^{j} \mathrm{e}^{\frac{t_i}{\tau_f}})}{\tau_f}} - \mathrm{e}^{-\frac{t-\tau_r ln(\sum_{i=0}^{j} \mathrm{e}^{\frac{t_i}{\tau_r}})}{\tau_r}}\right) \tag{B.45}$$

Which has the same form as B.41 but this time with constants:

$$\zeta = \tau_f ln(\sum_{i=0}^{j} \mathrm{e}^{\frac{t_i}{\tau_f}}) \tag{B.46}$$

$$\lambda = \tau_r ln(\sum_{i=0}^{j} \mathrm{e}^{\frac{t_i}{\tau_r}}) \tag{B.47}$$

in the same way of equation B.13 is generalized with B.20, equation B.41 generalizes with B.47.