MASTER THESIS

INTELLIGENT SYSTEMS

DESIGN SPACE EXPLORATION OF ASSOCIATIVE MEMORIES USING SPIKING NEURONS WITH RESPECT TO NEUROMORPHIC HARDWARE IMPLEMENTATIONS

ANDREAS STÖCKEL

Bielefeld University,
Faculty of Technology,
Cognitronics and Sensor Systems Group

SUPERVISED BY
PROF. DR.-ING. ULRICH RÜCKERT,
M.SC. THOMAS SCHÖPPING

DECEMBER 21, 2015 M16

The *Smart Thesis* template used in this document was written by Jan Philip Göpfert and Andreas Stöckel and is inspired by the *Classic Thesis* template developed by André Miede.

The source code of this document and all described software tools are available at: `https://github.com/hbp-sanncs/`

I slung the travelsack over my shoulder and cinched it tight across my back. Then I thumbed on my sympathy lamp, picked up the hatchet, and began to run.

I had a dragon to kill.

— KVOTHE IN "THE NAME OF THE WIND"

ABSTRACT

Artificial neural networks are well-established models for key functions of biological brains, such as low-level sensory processing and memory. In particular, networks of artificial spiking neurons emulate the time dynamics, high parallelisation and asynchronicity of their biological counterparts. Large scale hardware simulators for such networks – *neuromorphic* computers – are developed as part of the Human Brain Project, with the ultimate goal to gain insights regarding the neural foundations of cognitive processes.

In this thesis, we focus on one key cognitive function of biological brains, associative memory. We implement the well-understood Willshaw model for artificial spiking neural networks, thoroughly explore the design space for the implementation, provide fast design space exploration software and evaluate our implementation in software simulation as well as neuromorphic hardware.

Thereby we provide an approach to manually or automatically infer viable parameters for an associative memory on different hardware and software platforms. The performance of the associative memory was found to vary significantly between individual neuromorphic hardware platforms and numerical simulations. The network is thus a suitable benchmark for neuromorphic systems.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support of numerous individuals. First of all, I would like to thank Prof. Ulrich Rückert for the unique opportunity to contribute to the Human Brain Project, and Thomas Schöpping and Dr. Michael Thies for an introduction to the topic, guiding me along the right path and handing me valuable suggestions over the last year.

I would also like to dedicate my thanks to the researchers at the Electronic Visions group at the Kirchhoff-Institute for Physics in Heidelberg, the Advanced Processor Technologies Research Group at the University of Manchester, and everyone else involved in the neuromorphic hardware and software platforms for answering my multitudinous questions and beavering away on the problems I encountered.

On a technical side, none of this document would exist in this form without the work of thousands of volunteers who spent an astounding amount of time to contribute to the libre software projects used in writing and researching this thesis, including, but not limited to GCC, GNOME, the Linux kernel, Fedora, Octave, Matplotlib, Qt, LaTeX, Gimp and Inkscape. Thank you for helping to create a free and better world.

Furthermore, I wholeheartedly thank my parents for both the material and emotional support during my five-and-a-half-year exile in Bielefeld.

Last but not least, I express my sincere gratitude to all who read earlier revisions of this document and assisted me with their priceless feedback: Michael Thies, Thomas Schöpping, Benjamin Paaßen, Jan Philip Göpfert, Adriana Dreyer, Christoph Jenzen and Daniel Stöckel.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1 INTRODUCTION

Associative memories are believed to be one of the core mechanisms in human cognition: they allow us to classify sensory input in a fraction of a second and guide our thoughts along chains of semantic links [Pal13]. The Willshaw associative memory model (also Binary Neural Associative Memory, BiNAM) is one of the best understood, biologically plausible associative memory models [Ste61; WBL69; Pal80]. In conjunction with the neuromorphic hardware systems developed as part of the Human Brain Project (HBP) [Hum15b], an implementation of this model as spiking neural networks might provide a building block for the large-scale simulation of cognitive systems and serve as a benchmarking network which can be used to analyse the performance of said neuromorphic hardware. The goals of this thesis are to characterise the design space of a spiking Willshaw associative memory implementation and to develop measures and tools which can be used for both design space exploration and hardware benchmarking.

This chapter provides a high-level overview of this document: Section 1.1 elaborates on the motivation and goals sketched in the previous paragraph, Section 1.2 outlines the structure of the subsequent chapters and Section 1.3 closes with some remarks on notational conventions.

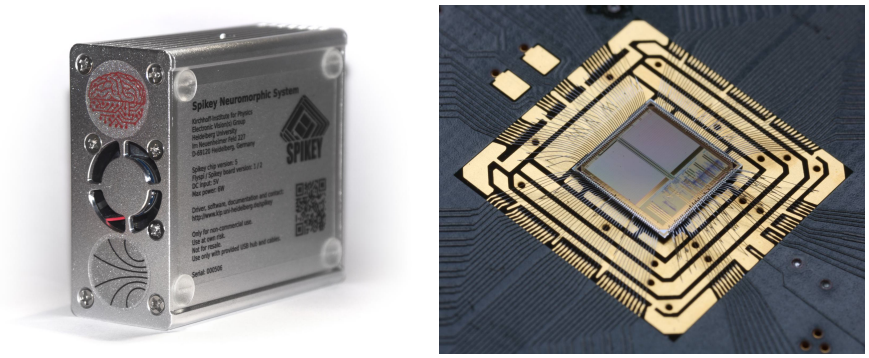
1.1 MOTIVATION AND GOALS

Here, we draw a sketch of the topical framework that encompasses this thesis, and, in doing so, present and motivate the overall goals. The following sections can only give a rough overview. Mentioned topics are discussed in greater detail in the following chapters – see the structure overview in Section 1.2 for more information.

1.1.1 *Neuromorphic hardware systems in the Human Brain Project*

The Human Brain Project (HBP) is a European research project which aims at advancing the understanding of the structure and function of the human brain at multi-level scales. A key goal of the project is the development of neuromorphic hardware systems: special purpose computers designed to enable the simulation of large, brain-like circuits and to provide scientists with means to validate hypotheses regarding brain function [Hum15a].

In contrast to the firing-rate coded artificial neural networks, which today are ubiquitous in the field of machine learning, the HBP neuromorphic platforms are based on spiking neural networks. Although still greatly simplified, spiking neural networks model processes in



(a) Spikey tabletop device (b) Spikey chip bonded to the circuit board

Figure 1.1: Photos of the *Spikey* neuromorphic hardware system developed by the Electronic Visions group at Heidelberg University. Size of the assembled device in (a) is about $8 \times 7 \times 3\,\text{cm}$; the picture of the actual chip in (b) is copied from `http://www.kip.uni-heidelberg.de/spikey`.

biological nervous systems more accurately than their firing-rate counterparts: neurons are represented as independent dynamic processes with asynchronous communication via electric pulses (spikes).

A more technical description of the neuromorphic systems can be found in Section 2.4.

The two neuromorphic hardware systems developed in the HBP possess fundamentally different architectures. The "many-core system" NM-MC1 built at the University of Manchester is a fully digital computer, constructed around the *SpiNNaker* chip [Fur+13]. In contrast, the "physical model" (NM-PM1) built at Heidelberg University, is an analogue-digital mixed signal system composed of entire silicon wafers of *HICANN* (High Input Count Analogue Neural Network) chips [Sch+10; Brü+11]. Both systems offer significantly faster simulation times for large spiking neural networks compared to software simulations on supercomputers, with NM-MC1 executing large networks at biological timescale and NM-PM1 with a speedup of 10 000 compared to biological timescale.

A third system – which is used in addition to the already mentioned ones – is the *Spikey* neuromorphic system (Figure 1.1). The Spikey chip at its core is a predecessor of the HICANN in NM-PM1. As such, it shares the same speedup factor of 10 000, but is limited to a single chip and features a less complex neuron model [Pfe+13]. Since Spikey has been in development for almost a decade now, both its hardware and software are in a relatively mature state.

1.1.2 *Willshaw associative memory as a spiking neural network*

Artificial neural networks in general and spiking neural networks in particular are discussed in Sections 2.1 to 2.3.

Due to their time dynamic and asynchronous nature, spiking neural networks are difficult to design: every neuron possesses a multitude of parameters, and small changes in a single neuron parameter can dramatically influence the behaviour of the network as a whole. This is already true for the most simple class of network topologies, so

called feed-forward networks, which do not allow recurrent connections (cycles in the network graph). On the other hand, it is reasonable to assume that biological neural networks have evolved with a robustness to neuronal variation. Indeed, the same neural circuits in two animals have been found to produce similar output on the network-level, although the intrinsic configuration of individual neurons varies significantly between the animals [PBM04; MT11].

As the Willshaw associative memory model merely is a theoretical concept, that was not designed with the perils of dynamical systems in mind, it describes no mechanism that would allow for the compensation of neuronal imprecisions. On the contrary, each output signal is produced independently by a single artificial neuron. There is no possible way in which coarsely estimated neuron parameters could be absorbed by network-level effects. For the transition of the theoretical memory architecture to a spiking neural network we could proceed in two different ways. By designing robust sub-networks for each theoretical neuron from which the desired behaviour emerges, or by literally translating the theoretical model to a spiking neural network and tuning the neuron parameters to precise values.

The Willshaw associative memory model is described in detail in Section 2.5.

We have decided to take the second approach. We aim at small and simple networks which only deviate slightly from the theoretical model, to ensure that the exhaustive theoretical results regarding the memory still apply. However, in order to find suitable neuron parameters, we need to provide an at least semi-automatic method which lets us choose "optimal" parameters for a given memory configuration.

1.1.3 *Associative memories as hardware benchmark*

The above decision on the spiking network architecture opens the door for another application. Since the network is scalable, possesses a highly regular and simple structure, and its theoretical behaviour under perfect conditions is well understood, it can serve as a hardware benchmark. Such a benchmark can be used to compare the performance of the individual platforms, and to quantify the influence of hard- and software changes.

Providing an assessable benchmarking network is important, as the HBP hardware platforms (excluding Spikey) are currently in an early stage of development and running even simple networks reliably on all platforms is likely to be enough of a challenge. Ideally, sweeps along multiple parameter axes of the design space can be performed. This would allow to find regions for which the system shows abnormal behaviour. For example, varying the memory size might uncover scaling issues.

1.1.4 *Goals of this thesis*

The top-level goal of this thesis is to provide a working Willshaw associative memory (BiNAM) implementation as spiking neural network which can be executed on the mentioned neuromorphic hardware systems. In order to fulfil this goal for varying memory configurations (e. g. size, data properties), we need to find a way to explore the network design space. This requires that we have defined the design space itself and performance measures we can assign to every point in the space.

Unfortunately, and unsurprisingly, the design space is high-dimensional, which prevents any exhaustive exploration. Therefore, we intend to develop and evaluate estimations of the network performance. These should allow an interactive exploration of a two-dimensional projection of the design space and help to restrict the parameter space to regions, in which time-consuming network simulations (potentially accelerated by the neuromorphic hardware) are sensible. Additionally, it should be possible to use the performance estimations for automatic parameter optimisation.

These goals also come with a significant engineering task, as the software tools for interactive design space exploration, parameter optimisation and execution on the hardware platforms have to be implemented. Finally, we have to investigate whether the built tools can be reliably used for benchmarking.

1.2 STRUCTURE

Whereas Section 1.1 approached the topics in this thesis from a bird's-eye perspective and in no particular order, this section linearly trudges along the chapters and traces the golden thread which guides through the pages to come.

In Chapter 2, we start by summarising the related work that sits at the foundation of this thesis: we touch the relevant neurobiological basics, introduce notable neuron and neural network models, and present the neuromorphic hardware platforms and software simulators. Finally, the Willshaw associative memory model (BiNAM) is described and compared to other models.

In order to transition the theoretical BiNAM model to a spiking neural network, Chapter 3 defines a parametrised set of spiking BiNAM implementations. Combined with the neuron model parameters, these implementation parameters span the associative memory design space we seek to explore. Various associative memory evaluation measures are then proposed, which allow the assignment of a set of scores to any given point in the design space. As a side-effect, these measures allow comparison – and eventually benchmarking – of different hardware and software platforms. The chapter concludes with thoughts on

BiNAM test data generation. In its entirety, the material in Chapters 2 and 3 allows the construction of a complete BiNAM design space exploration pipeline.

Armed solely with the methods described beforehand, however, any even rudimentarily exhaustive design space exploration would be infeasible, regardless of neuromorphic hardware acceleration. Thus, Chapter 4 takes a step back and describes BiNAM performance estimates of varying complexity, based on single neuron simulation. To achieve the fastest possible execution speed, we compare the performance of various numerical differential equation integration methods. Finally, a method for fractional neuron output spike count estimation is presented, which in conjunction with a naive Downhill-Simplex algorithm effectively allows automated neuron parameter optimisation with respect to a given set of network parameters. The presented work can be used to limit the parameter space to interesting regions, which can then be explored by means of expensive full network simulations.

Such simulations are conducted in Chapter 5, which describes experiments testing the evaluation measures defined in Chapter 3 on the neuromorphic hardware platforms. The hardware results are then compared to software simulations and the coarse estimates from the fourth chapter.

Finally, Chapter 6 summarises the insights obtained and lists possible future work that was out of scope for this thesis.

1.3 NOTATIONAL CONVENTIONS

This section informally lists some important notational conventions employed throughout the thesis.

Margin notes contain additional remarks or small sketches which aim at providing a better understanding of the material – however, they can be safely skipped; all relevant information is presented in the main text.

Symbols Great care has been taken to consistently impose a single meaning on most mathematical symbols. Exceptions to this rule are "local variables", including – but not limited to – the symbols i, j, k, ℓ which are used as generic indices, for example as summation indices, loop counters or to point at a generic element of a set, tuple, vector or matrix. The meaning of symbols used in more than one occasion can be looked up in the symbol overview in the appendix.

Vector and matrix indices Vectors are marked as such with a vector arrow and are generally assumed to be column vectors. If individual matrix or vector elements are accessed, this is denoted as $(\vec{x})_i$ (the i-th component of $\vec{x}$) or $(M)_{ij}$ (the element in the i-th row and j-th column of the matrix M) respectively. An exception to this rule are algorithms in pseudo-code where individual vector and matrix elements are accessed in the "square brackets"-notation, for example $\vec{x}[i]$ and $M[i, j]$. Regardless of the notation, vector and matrix indices follow the mathematical convention and always start with one.

Sets and tuples Sets are usually typeset in fraktur (e. g. $\mathfrak{b}$, $\mathfrak{B}$). Double-struck letters (e. g. $\mathbb{B}, \mathbb{N}, \mathbb{R}$) denote ranges of numbers. The operator "$\|$" denotes the concatenation of two tuples: let $a = (a_1, \ldots, a_i)$ and $b = (b_1, \ldots, b_j)$ denote two sequences of length $|a| = i$ and $|b| = j$. The operation $a \| b$ is then defined as:

$$a \| b = (a_1, \ldots, a_i, b_1, \ldots, b_j) \tag{1.1}$$

Discontinuities in differential equations Spiking neural network models are generally formulated as differential equations. However, they contain discontinuities which often are expressed in the literature with the help of the Dirac delta $\delta(t)$:

$$\int_{-\infty}^{\infty} f(t) \cdot \delta(t) \, \mathrm{d}t = f(0) \,. \tag{1.2}$$

Use of the Dirac delta may facilitate mathematical analysis, but in the opinion of the author tends to obscure the actual concept that is being described. We therefore use a less mathematical notation which involves the "$\leftarrow$" (read "gets") operator. For example, a differential equation with a discontinuity at time t_0 is described as

$$\dot{u}(t) = g \cdot (u(t) - u_0) \,, \tag{1.3}$$

$$u(t) \leftarrow u(t) + \Delta u \quad \text{if } t = t_0 \,, \tag{1.4}$$

and supposed to be equivalent to the more "correct" mathematical notation

$$\dot{u}(t) = g \cdot (u_0 - u(t)) + \delta(t - t_0) \cdot \Delta u \,. \tag{1.5}$$

2 BACKGROUND AND RELATED WORK

> *The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.* [...] *We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.*
>
> — MCCARTHY ET AL., 1955, PROPOSAL FOR THE DARTMOUTH CONFERENCE

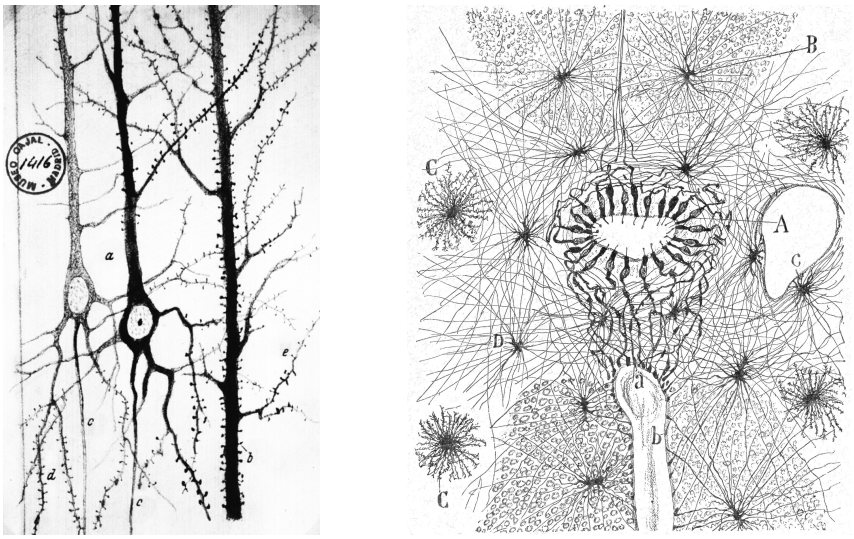
The primary goal of this thesis is to implement an associative memory model as a spiking neural network on top of neuromorphic hardware. In this chapter we aim at providing a terse summary of the mentioned fields: Sections 2.1 to 2.3 address artificial neural network models in general, the neurobiological concepts inspiring spiking neural networks, relevant spiking neuron models, and their parameters. Section 2.4 focuses on the neuromorphic hardware systems in the HBP. In Section 2.5 we discuss the notion of associative memories and expand on the Willshaw associative memory model relied upon in this thesis.

2.1 HISTORY OF ARTIFICIAL NEURAL NETWORK MODELS

In 1780 Luigi Galvani discovered that the injection of electric potentials into animal muscle tissue causes contractions. He was the first to notice that electricity could play a role in the animation – or liveness – of animals and laid the groundwork for research concerning electrophysiology [Pic97].

Sixty-eight years later, in 1848, Emil du Bois-Reymond discovered discrete electrical pulses generated by nerve cells [Pea01]. It took another seventeen years until Julius Bernstein (supported by du Bois-Reymond) could successfully record one of these *action potentials* or *spikes* on paper. Today we know that action potentials are the primary way of encoding information in the nervous system [Sch83].

At the end of the nineteenth century, technological advances in microscopy and a new staining method invented by Camillo Golgi allowed scientists to examine individual neurons in brain and spinal tissue samples (Figure 2.1). In 1887 Santiago Ramón y Cajal was the first to propose neurons as distinct base units of information processing in biological systems. For their work Golgi and Cajal received the 1906 Nobel prize. Their discoveries gave rise to the *neuron doctrine*, the idea that spinal cord and brain are made of basic building blocks – neurons – and their support structures [Gli06].



(a) Pyramid cells (b) Neurons in spinal marrow

Figure 2.1: Neural tissue prepared using Golgis method and drawn by Santiago Ramón y Cajal around 1900. (a) shows pyramidal neurons in brain tissue, (b) neurons of varying shape in the white spinal marrow substance [Caj04].

As the understanding of the neurobiological mechanisms advanced, another question began to dawn in the scientific world: if animal – and human – behaviour was solely determined by the electrophysiological properties of neural networks, could it not be possible to build machines that simulate processes in the brain up to cognition and intelligence? In the 1940s researchers began to construct mathematical models which mimic structural properties of biological neural networks. The development of artificial neural networks since then can be broken down into three distinct generations [Maa97].

2.1.1 *First generation: binary McCulloch-Pitts cells*

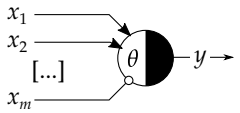


Figure 2.2: Sketch of a McCulloch-Pitts neuron: excitatory (arrow) and inhibitory (circle) inputs are accumulated. If a threshold θ is passed, the output y is set to one, zero otherwise.

In 1943 Warren McCulloch and Walter Pitts proposed the first artificial neural network model. In order to cope with the high diversity and complexity of biological neurons, their model is based on several simplifying assumptions: the nervous system is built of a network of neurons, each consisting of a cell body (*soma*) and an axon. They gather input from connected neurons through excitatory or inhibitory synapses located at dendritic extensions of the soma (*dendrites*). If the excitation of a neuron passes a threshold, the neuron responds with a binary "all-or-none" spike, that travels along the axon to other neurons, where it is received as input (Section 2.2 and Figure 2.3). Furthermore, McCulloch and Pitts argue that transmission of signals along the axon is almost instantaneous and considerable delay occurs only at the synapses. This allows to disregard spike times and instead

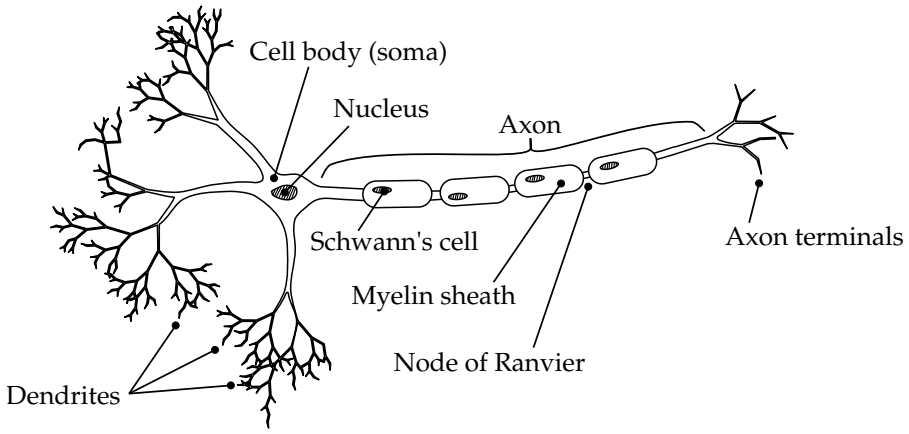


Figure 2.3: Sketch of a schematised biological model neuron. Input spikes arrive at synapses located at the dendrites and are processed in the cell body. Resulting output spikes travel along the axon to the axon terminals, which connect to other neurons or a neuromuscular junction. Inspired by [Kan+12].

synchronously propagate binary values between neurons in discrete time steps [MP43].

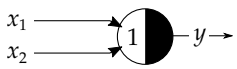
Mathematically, a single McCulloch-Pitts cell with binary input vector $\vec{x} = (x_1, \ldots, x_m)^\top \in \mathbb{B}^m$, output $y \in \mathbb{B}$, and $\mathbb{B} = \{0, 1\}$ can be described as follows (Figure 2.2)

$$y = H(\vec{w}^\top \cdot \vec{x} - \theta) \qquad \text{where} \quad H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.1}$$

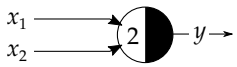
The weights $\vec{w} = (w_1, \ldots, w_m)^\top \in \{-1, 1\}^m$ model excitatory ($w_i = 1$) or inhibitory ($w_i = -1$) synaptic connections to the input x_i, the threshold $\theta \in \mathbb{Z}$ describes the minimum excitation that causes a "one" as output. The function $H(x)$ is also called "Heaviside step function".

As shown in Figure 2.4, the cells can be used to construct the basic operators of Boolean algebra. It follows that any computable function can be described by a large network of McCulloch-Pitts cells – they are Turing complete [CP96].

OR ($y = x_1 \vee x_2$):



AND ($y = x_1 \wedge x_2$):



NOT ($y = \neg x$):

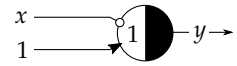


Figure 2.4: McCulloch-Pitts artificial neuron as Boolean operators.

2.1.2 *Second generation: firing-rate coded neural networks*

In 1958 Frank Rosenblatt extended the binary McCulloch-Pitts cell to the so called "perceptron". Weights w_i and neural input x_i in Equation (2.1) are now real-valued instead of binary. Biologically, this change can be motivated by the observation that some neurons operate in a mode known as "tonic spiking" in which they output discrete spikes at a certain rate that monotonously depends on the excitation of the neuron (Figures 2.11(a) and 2.11(g)). The real valued input $\vec{x} \in \mathbb{R}^m$ can be interpreted as the average firing-rate of the pre-synaptic neuron, the

weights $\vec{w} \in \mathbb{R}^m$ describe the influence of a synapse on the excitation of the neuron [Hay11].

Rosenblatt's most important contribution, though, is a learning rule which allows to train the weights $\vec{w}$ in such a way, that the perceptron outputs a desired answer y_k for a certain input $\vec{x}_k$, allowing to solve linear classification and regression tasks [MP87].

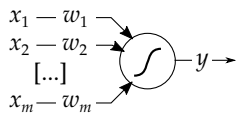


Figure 2.5: Sketch of a firing-rate coded artificial neuron. Each neuron in the network computes the weighted sum of inputs $x_1, \ldots, x_m$, applies a non-linearity f and outputs an activation value y that might be fed into other neurons or act as part of the network output.

In the context of the feed-forward multilayer perceptron (MLP), the neuron model is generalised to the "firing-rate artificial neuron", replacing the Heaviside function H with an arbitrary non-linear, sigmoid function f and fusing the threshold θ as an additional dimension ("bias") into the input $\vec{x}$ and the weights $\vec{w}$ (Figure 2.5)

$$y = f(\vec{w}^\top \cdot \vec{x}) \,. \tag{2.2}$$

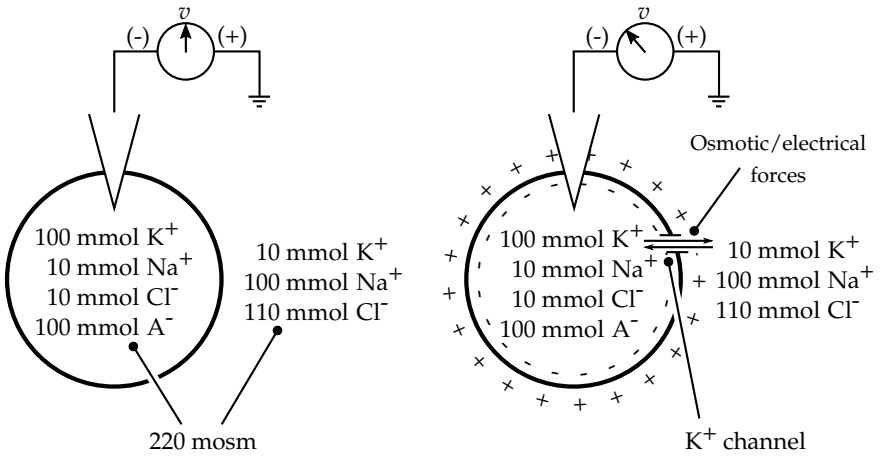
The weights $\vec{w}$ of the individual neurons in a MLP can be easily trained using the back-propagation algorithm (a generalisation of Rosenblatt's perceptron learning rule). Today, with the broad availability of massively parallel computing hardware, large MLPs with many hidden layers are employed with success in the field of "deep learning" [HOT06].

2.1.3 *Third generation: spiking neural networks*

McCulloch and Pitts assumed that coarse, discrete timesteps are sufficient for the propagation of neural output – a paradigm that is adopted by second generation networks. Experiments suggest however, that exact spike timing and spike time correlation within neuron populations are used to encode information in the nervous system [SN94]. At the end of the 1980s, these discoveries gave rise to the third generation of neural networks, in which neurons are simulated as dynamical systems with asynchronously generated binary spikes [Maa97].

Besides being closer to biology, spiking neural networks have several practical advantages over their predecessors: whereas firing-rate models require the transfer of the neuron state of every neuron in every time step, the asynchronicity of spiking networks only requires communication whenever a neuron generates a spike. Due to the loose coupling of individual neurons, spiking networks lend themselves to be simulated energy efficiently on massively parallel, asynchronous hardware. Furthermore, given their time-dynamic nature, spiking neural networks intrinsically process time series of data.

On the other hand, simulation of neuron time dynamics is computationally intensive, training of spiking networks is more complicated compared to their firing-rate counterpart and – at least for simple models – biological plausibility is still limited: usually neither spatiality, nor the influence of neuromodulators are simulated. Nevertheless, spiking neural networks may provide a useful simulation platform for entire brain circuits [JL07] and may in the future allow to simulate deep networks on specialised, energy efficient hardware [HM13; Sch15].



(a) Membrane without K^+-channel (b) Added K^+-channel

Figure 2.6: Ion compositions in intra- and extracellular space and with closed (a) and opened (b) selectively permeable ion channel. See Section 2.2.1 for a description. Adapted from [Kan+12].

We continue with a description of the spiking neural network models this thesis is based on in Section 2.3, but before, in Section 2.2, we quickly explore their biological basis.

2.2 BIOPHYSICAL NEURON MODEL

Biological observations of neuronal behaviour have (amongst others) been captured in the biophysically meaningful Hodgkin-Huxley (HH) neuron model, introduced in 1952 [HH52]. It builds the basis of most simplified spiking neural network models in neuroinformatics. In the remainder of this section we discuss the relevant parts of this model.

2.2.1 *Passive electrophysiological properties of the neuron membrane*

Ions in the intra-/extracellular fluid which usually play a role in neurobiological processes are: sodium ions Na^+*, chloride* Cl^-*, potassium ions* K^+*, calcium ions* Ca^{2+}*, as well as negatively charged amino acids* A^-*.*

The electrical properties of a single neuron are caused by a different ion compositions in intra- and extracellular fluid, and selectively permeable ion channels in the cell membrane. When measuring the electrical potential between the intra- and extracellular space – the *membrane potential* u – of an inactive neuron, one finds the intracellular space being more negatively charged than the extracellular space [Kan+12]. This particular voltage is called the *resting* or *leak* potential E_L.

An examination yields differences in the intra- and extracellular fluid ion-composition, with the intracellular composition being maintained by ion pumps in the cell membrane. However, seemingly contradictory to the previous result, both fluids are electrically neutral and have the same osmotic concentration. Only if the membrane was a perfect insulator – as assumed in Figure 2.6(a) –, there would be no measurable potential.

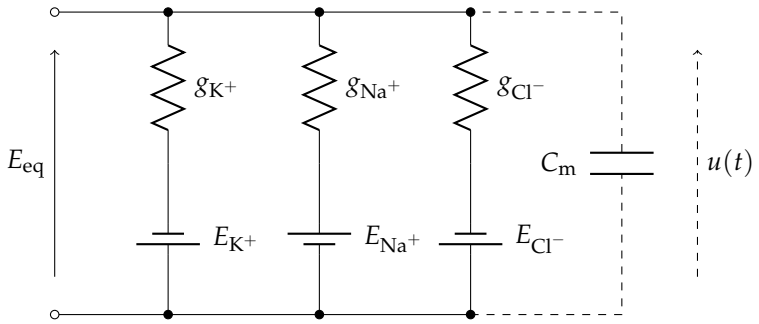


Figure 2.7: Model equivalent circuit diagram of the neuron membrane for three ion channels. The potential that can be measured across the terminals on the left (without the capacitor) corresponds to the equilibrium potential as described in equation Equation (2.4). Introducing a capacitor (dashed) with capacitance C_m transforms the circuit into the time-dynamic neuron base model in Equation (2.5).

The number of ions involved in the generation of the equilibrium potential is negligibly small compared to the total number of ions in the cell.

Experiments show, that the membrane contains selectively permeable ion channels. In its resting state, the membrane is mostly permeable for potassium ions K^+. Due to the difference between intra- and extracellular ion concentration, an osmotic force causes K^+ to flow out of the cell. The total ionic current is proportional to the number channels in the membrane, or its *permeability* for K^+. Missing K^+ cause the intracellular fluid to become slightly negatively charged, creating a countering electric force on the positively charged ions. As sketched in Figure 2.6(b), the system converges towards an equilibrium state with potential E_{Eq} in which osmotic and electric force are equal.

Consider the equilibrium potential $E_{\mathcal{I}}$ for a membrane that is permeable for an ion species $\mathcal{I}$ only. We refer to $E_{\mathcal{I}}$ as the *reversal potential* for $\mathcal{I}$: its ionic current reverses at this potential. For intra- and extracellular ion concentrations $[\mathcal{I}]_{in}$ and $[\mathcal{I}]_{out}$, $E_{\mathcal{I}}$ is given according to the Nernst equation as

$$E_{\mathcal{I}} = \frac{R \cdot T}{z \cdot F} \cdot \ln\left(\frac{[\mathcal{I}]_{out}}{[\mathcal{I}]_{in}}\right), \tag{2.3}$$

where R is the ideal gas constant, T the temperature in Kelvin, F Faraday's constant, and z the ion charge in elementary charges [Kan+12]. Given the (relative) permeabilities or *conductances* $g_{\mathcal{I}}$ of the cell membrane for each ion species $\mathcal{I}$, the equilibrium potential E_{eq} can be calculated according to the Goldman–Hodgkin–Katz equation as

$$E_{eq} = \frac{\sum_{\mathcal{I}} g_{\mathcal{I}} \cdot E_{\mathcal{I}}}{\sum_{\mathcal{I}} g_{\mathcal{I}}} = \frac{\sum_{\mathcal{I}} g_{\mathcal{I}} \cdot E_{\mathcal{I}}}{g_{tot}}. \tag{2.4}$$

Correspondence between Fig. 2.7 and Eq. (2.4) can be easily shown using Kirchhoff's laws.

The behaviour modelled by Equation (2.4) corresponds to an electric circuit, consisting of parallel voltage sources with voltage $E_{\mathcal{I}}$ and a series resistor with conductance $g_{\mathcal{I}}$ for each ion channel (Figure 2.7).

Due to inertia in the system, the membrane potential adapts slowly to any change, for example changes in the ion channel permeabilities.

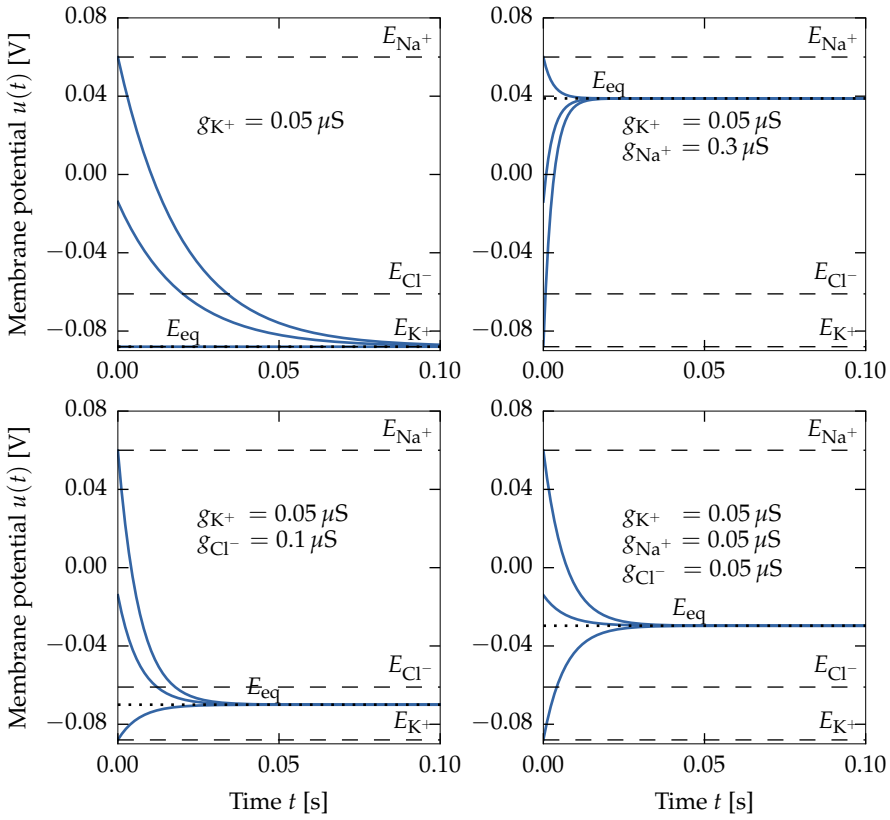


Figure 2.8: Membrane potential over time for varying membrane conductances, according to Equation (2.5). The reversal potentials are set to $E_{K^+} = -88\,\text{mV}$, $E_{Na^+} = 61\,\text{mV}$ and $E_{Cl^-} = -60\,\text{mV}$, the membrane capacitance C_m to 1 nF.

As depicted in Figure 2.7, this time-dynamic can be modelled by adding a capacitor with capacitance C_m – the *membrane capacitance* – in parallel to the equivalent circuit.

The time differential of the voltage across the capacitor $\dot{u}(t)$ is proportional to the current $i(t)$, which in turn is proportional to the difference $u(t) - E_{eq}$. Hence, the circuit can be described as a linear differential equation

By convention, positive currents $i(t)$ drive the membrane potential towards more negative values, negative currents towards more positive values.

$$-C_m \cdot \dot{u}(t) = i(t) = g_{tot} \cdot (u(t) - E_{eq}) = \sum_{\mathcal{I}} g_{\mathcal{I}} \cdot (u(t) - E_{\mathcal{I}})\,. \quad (2.5)$$

Figure 2.8 shows the system for varying u_0 and different membrane conductances. In all four plots the membrane is always slightly permeable for potassium ions K^+. If the membrane is not permeable for any other ion, this causes $u(t)$ to slowly converge towards E_{K^+}. The velocity of the convergence is proportional to g_{tot}. Permeability for chloride or sodium ions pulls $u(t)$ towards their reversal potential.

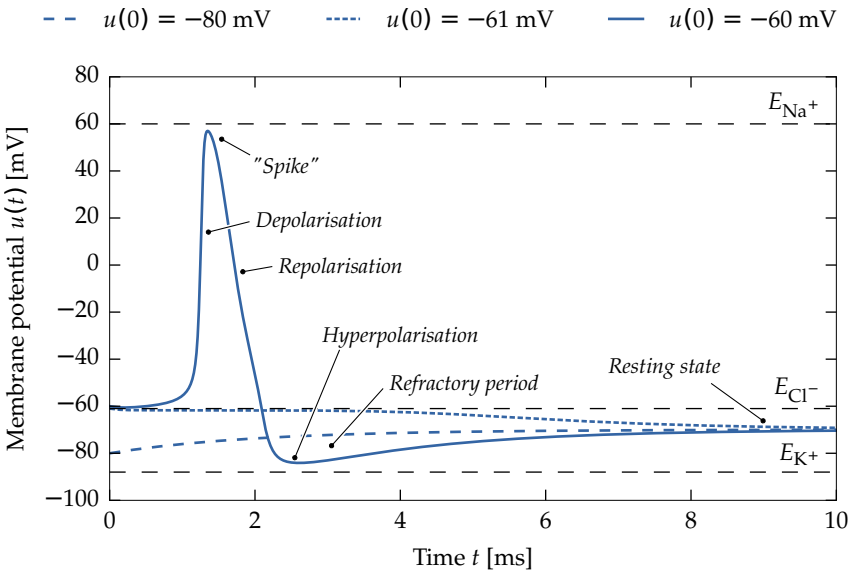


Figure 2.9: Annotated sketch of an action potential as produced by the Hodgkin-Huxley (HH) model. The membrane potential of three neurons is clamped to certain membrane potentials $u(t)$ at $t = 0$. If a certain threshold potential is reached, the neuron generates an action potential; below this potential behaviour similar to a passive membrane can be observed.

2.2.2 *Action potentials*

A passive cell membrane is not sufficient to explain action potentials and thus lacks an integral part of spiking neural networks – the spikes. As soon as the neuron membrane potential surpasses a certain threshold E_{Th}, the neuron will suddenly depolarise up to a value E_{spike}, followed by a decrease below the resting potential E_{L} (hyperpolarisation) to the reset potential E_{reset}. The neuron stays close to the reset potential for a certain time span (known as "refractory period"), until the membrane potential again converges towards E_{L} (Figure 2.9).

Ion channels are binary: they can either be open or closed – the HH model therefore describes the ion channel state probabilistically over a population of channels as three state variables in addition to the membrane potential.

Mechanistically, this behaviour is produced by voltage-gated sodium and potassium ion channels (Na^+, K^+): the probability of open Na^+ channels increases with the membrane potential, causing a positive feedback loop and a depolarisation of the membrane up to E_{spike}. Meanwhile, the voltage-gated channels for K^+ open with the same mechanism, albeit a little slower, and the Na^+ channels transition into a closed and deactivated state, causing the sudden re- and hyperpolarisation. Facilitated by the hyperpolarisation the Na^+ and K^+ channels reset to their initial state, allowing the generation of new action potentials [Kan+12].

An evolutionary (ultimate) explanation of action potentials is their suitability for signal propagation along the axon. As the neuron membrane is neither a perfect insulator nor the intracellular fluid a good conductor, pure electric signals are dampened with increasing spa-

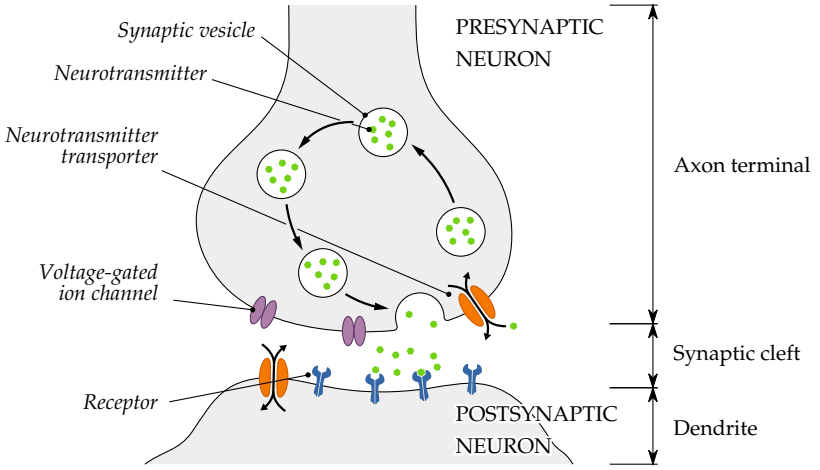


Figure 2.10: Chemical synapse, adapted from `https://commons.wikimedia.org/wiki/File:SynapseSchematic_en.svg`. See text for description.

tial distance. "All-or-none" spikes on the other hand allow constant signal renewal without information loss: as depicted in Figure 2.3, portions of the axon are insulated with *Myelin* (decreasing the leak-conductance and thus the potential gradient), allowing fast but lossy electrical propagation of the signal. The Myelin sheath is regularly interrupted by *Nodes of Ranvier*, where the neuronal ion-channel action potential generation mechanism renews the action potential [Kan+12].

Spiking signals in biology can be explained with similar rationale as digital representations in computers: discrete signals can recover from noise without information loss, whereas analogue signals are irrecoverably altered.

2.2.3 *Chemical synapses*

As already mentioned in the discussion of artificial neural network models (Section 2.1), synapses are the basis for inter-neuron communication and thus neural networks. There are two types of biological synapses: electrical synapses, which allow for a direct exchange of intracellular fluid, and the more common chemical synapses, sketched in Figure 2.10 and described in the following. If an action potential arrives at the axon terminal of the presynaptic neuron, vesicles containing a neurotransmitter fuse with the cell membrane and release the transmitter into the synaptic cleft. The transmitter then docks onto receptors located at the dendrites of the postsynaptic neuron, where they – depending on the configuration of the dendritic part of the synapse – trigger the opening or closing of ion channels and either excite or inhibit the neuron (pull the membrane towards more positive or negative potentials) [Kan+12].

Compared to the spike transmission along the axon, the delay occurring at the synapse is rather large. The release of a neurotransmitter furthermore low-pass filters the incoming spikes, stretching their effect over longer time-periods.

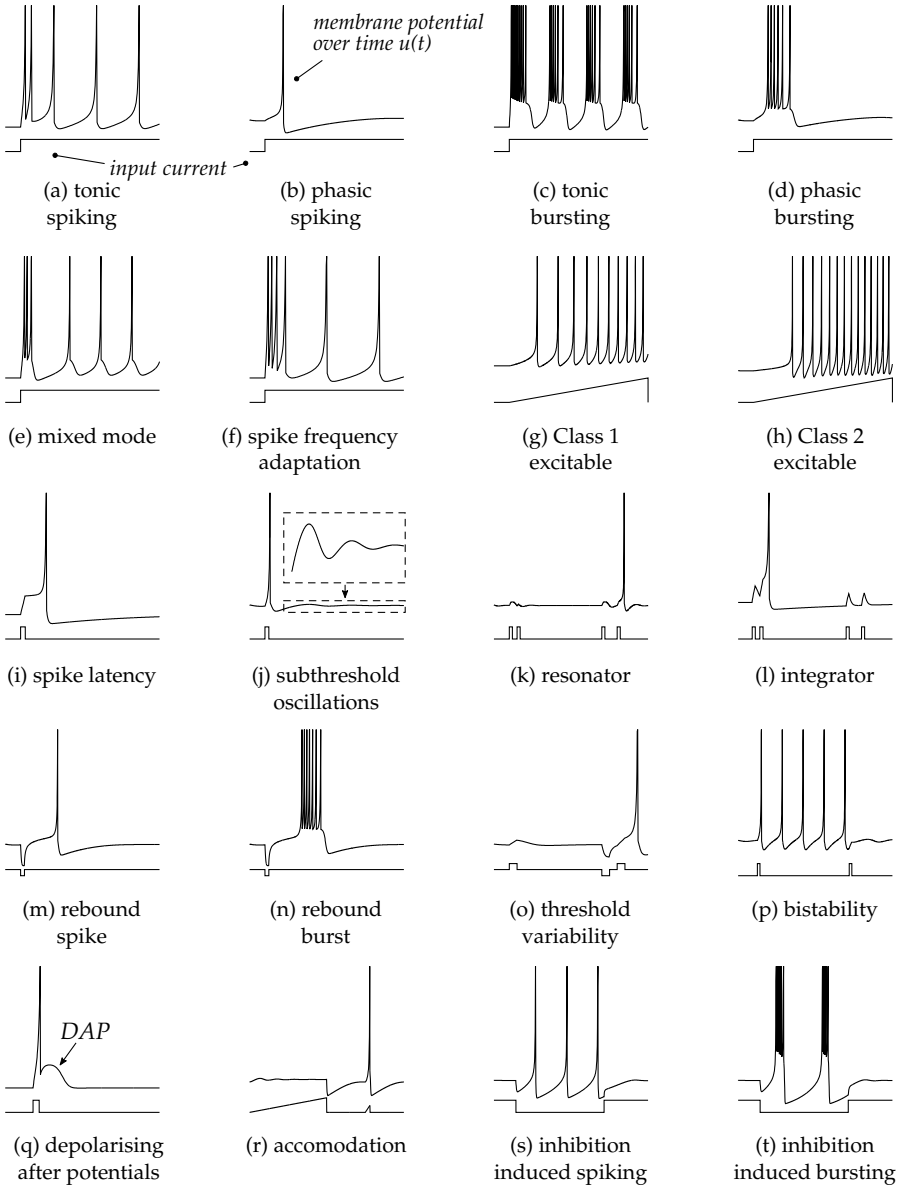


Figure 2.11: Sketches of spiking neuron behavioural patterns. Each subgraph shows membrane potential $u(t)$ and input current $I_{\text{syn}}(t)$ over time. See [Izh04] for more information. Electronic version of the figure and reproduction permissions are freely available at `http://www.izhikevich.com/`.

2.3 SIMPLIFIED NEURON AND SYNAPSE MODELS

The previous sections sketched two extremes: the biophysically meaningful Hodgkin-Huxley (HH) model, and the simplistic firing-rate neuron models used in machine learning. While it is surely possible to use the HH model as the underlying neuron model for artificial spiking neural networks, its evaluation is computationally expensive [Izh04] and mathematical analysis is complicated due to its intricate dynamics.

To overcome these limitations, less complex neuron models have been developed. However, their reduced complexity often comes at the cost of reduced expressiveness: Figure 2.11 shows the variety of neuron behaviours observed in biological neurons that – given the correct neuron parameters – can be described with the HH model. Yet, the simplest neuron models only support basic modes of operation (e. g. "tonic spiking"). In the remainder of this section we introduce the synapse and neuron models used on the HBP neuromorphic hardware platforms, list their parameters, and discuss their expressiveness. More information on spiking neuron models can be found in [GK02].

As discussed in [Izh04], the HH model requires up to two magnitudes more floating point operations for the same timespan as comparably expressive, but simpler models.

2.3.1 *Neuron model base equation*

The representation of the cell membrane as a capacitor with capacitance C_m is the conceptual basis of most spiking neuron models. Over time the capacitor is charged and discharged by two currents: the intrinsic channel current $I_\mathrm{chan}(u,t)$, corresponding to the sum of ionic currents through the ion channels, and the synaptic – or external – current $I_\mathrm{syn}(u,t)$ modelling the ionic currents in the synapses as response to external input

$$-C_\mathrm{m} \cdot \dot{u}(t) = i(t) = I_\mathrm{chan}\left(u(t),t\right) + I_\mathrm{syn}\left(u(t),t\right) . \tag{2.6}$$

A summary of the relevant neuron model parameters is given in Table 2.1 on page 23.

The form of the channel current I_chan depends on the concrete neuron model, whereas the synaptic current I_syn is determined by the synapse model.

2.3.2 *Synapse models*

Generally, the synaptic current I_syn is the sum of currents induced by all synapses of a neuron (the number of synapses equals the fan-in of the neuron in the network)

$$I_\mathrm{syn}(u,t) = \sum_k I^k_\mathrm{syn}(u,t) . \tag{2.7}$$

The synapse model is usually independent of the neuron model – synapses are solely a biologically inspired way to inject a current into a neuron. However, the boundaries will become fuzzy as we introduce excitatory and inhibitory synapses.

The current I^k_syn caused by each individual synapse k is determined by the internal synapse state. This state is modified whenever the synapse receives a pre-synaptic spike and steadily converges to a resting value. The amplitude of the modification depends on the synapse weight w_k. The physical unit of w_k depends on the synapse model. Two synapse models are common: *current*-based and *conductance*-based synapses.

Current based synapses with exponential decay Synapses of this type do not possess additional state variables – their sole state is the current I^k_syn itself. This model is particularly interesting as I^k_syn does not depend on u, which in some cases enables fully analytical solutions of the differential in Equation (2.6). Whenever an input spike is received

at time t, the current is increased by w_k (in ampere). The synaptic current then exponentially decays to zero over time with time constant τ_k, modelling the low-pass behaviour mentioned in Section 2.2.3. A single current based synapse k can be described as

$$I^k_{\text{syn}}(t) \leftarrow I^k_{\text{syn}}(t) + w_k \quad \text{on spike for } k \text{ at } t \tag{2.8}$$

$$-\mathrm{d}/\mathrm{d}t\ \tau_k \cdot I^k_{\text{syn}}(t) = I^k_{\text{syn}}(t)\,. \tag{2.9}$$

Conductance based synapses with exponential decay This synapse model is biologically more plausible as it models the transmitter gated membrane channels in biological synapses to a certain extent. As it is available on all neuromorphic hardware platforms, it is the model of choice in this thesis. In contrast to the current based channel, each synapse has a conductance g_k as internal state. An input spike at time t increases the conductance by w_k (in siemens). As with the current based model, the state variable decays with the synapse-specific time constant τ_k.

$$g_k(t) \leftarrow g_k(t) + w_k \quad \text{on spike for } k \text{ at } t \tag{2.10}$$

$$-\mathrm{d}/\mathrm{d}t\ \tau_k \cdot g_k(t) = g_k(t) \tag{2.11}$$

The actual synaptic current I^k_{syn} depends on the state g_k, the current membrane potential u and the synaptic channel reversal potential E_k.

$$I^k_{\text{syn}}(u,t) = g_k(t) \cdot (u - E_k) \tag{2.12}$$

Note that the dependency of I^k_{syn} on u renders finding a closed form solution of the neuron differential equation impossible for any practically useful channel current equation $I_{\text{chan}}(u,t)$. An example synapse conductivity trace over time with incoming pre-synaptic spikes is shown in Figure 2.13.

2.3.3 *Excitatory and inhibitory synapses*

In the software interfaces, the synapse weight w can be chosen individually per synapse. Usually, positive w indicate excitatory synapses, negative w inhibitory synapses (with weight $|w|$).

In theory, the reversal potential E_k and time constant τ_k could be chosen individually for each conductance based synapse. This is biologically implausible, as the reversal potentials are defined by the fixed ion concentration gradients for K^+, Na^+ and Cl^-. Most simulators – including the neuromorphic hardware systems – restrict the number of different synapse types per neuron to two: an excitatory synapse with parameters E_e, τ_e (corresponding to the Na^+ ion channels), and an inhibitory synapse with parameters E_i, τ_i (corresponding to the K^+ ion channels).

Usually, the excitatory reversal potential is chosen as $E_e \geq E_{\text{Th}}$: input spikes that arrive at an excitatory synapse push the membrane potential u towards the threshold potential E_{Th} and enable the generation of output spikes. Analogously, the inhibitory synapse reversal

potential is chosen as $E_\mathrm{i} \leq E_\mathrm{L}$, allowing spikes reaching inhibitory synapses to hyperpolarise the neuron.

The restriction of the number of synapse types simplifies the equation for I_syn: two state variables g_e and g_i have to be stored per neuron and I_syn can be written as

$$I_\mathrm{syn}(u,t) = g_\mathrm{e}(t) \cdot (u - E_\mathrm{e}) + g_\mathrm{i}(t) \cdot (u - E_\mathrm{i})\,, \tag{2.13}$$

where g_e or g_i are adapted according to Equation (2.10) for input spikes reaching excitatory/inhibitory synapses. The conductances decay with time constants τ_e and τ_i as described in Equation (2.11).

2.3.4 *Linear integrate-and-fire neuron model*

The linear integrate-and-fire (LIF) neuron model can be seen as a minimal extension of Equation (2.6): the simulated neuron membrane contains a leak channel with constant conductance g_L, pulling the membrane towards the resting potential E_L. For excitatory and inhibitory synapses as described in Section 2.3.3, the differential equation for the membrane potential $u(t)$ is given as

$$\begin{aligned} -C_\mathrm{m} \cdot \dot{u}(t) &= g_\mathrm{L} \cdot (u(t) - E_\mathrm{L}) + I_\mathrm{syn}(u(t), t) \\ &= g_\mathrm{L} \cdot (u(t) - E_\mathrm{L}) + g_\mathrm{e}(t) \cdot (u(t) - E_\mathrm{e}) + g_\mathrm{i}(t) \cdot (u(t) - E_\mathrm{i})\,. \end{aligned} \tag{2.14}$$

The output action potential is not explicitly formed as a spike in the LIF model. For visualisation purposes a spike reaching up to a potential E_spike is often artificially inserted at the threshold-crossing.

The above equation does not account for spike generation and refractoriness of the neuron. An output spike is generated, whenever the membrane potential $u(t)$ crosses a certain threshold $E_\mathrm{Th} > E_\mathrm{L}$. The refractory period is modelled by tracking the last output spike time t_spike (initialised with $-\infty$). While the condition $t - t_\mathrm{spike} \leq \tau_\mathrm{ref}$ holds, the membrane potential is clamped to the reset potential $E_\mathrm{reset} \leq E_\mathrm{L}$

$$t_\mathrm{spike} \leftarrow t \qquad \text{if} \qquad u(t) \geq E_\mathrm{Th} \tag{2.15}$$

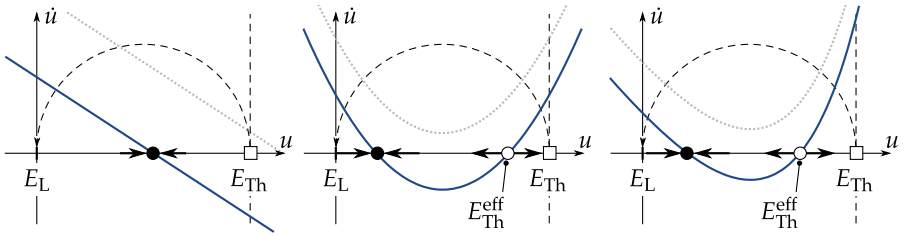
$$u(t) \leftarrow E_\mathrm{reset} \qquad \text{while} \qquad t - t_\mathrm{spike} \leq \tau_\mathrm{ref}\,. \tag{2.16}$$

The expressiveness of this model is severely limited: given a constant input current $I_\mathrm{syn}(t)$, the model can only operate in the tonic spiking mode (Figure 2.11). All state information is lost once an output spike is issued and the membrane potential is reset [Izh04].

More complex behaviour such as bursting can only be realised in conjunction with the synapse model. In combination with conductance based synapses, the LIF model is also referred to as IfCondExp model. Despite its shortcomings the model is extensively used in spiking neural network simulations. Furthermore, it is supported by all neuromorphic hardware platforms in the Human Brain Project (HBP).

2.3.5 *Non-linear integrate-and-fire models*

The LIF neuron model has a severe instability in its dynamics: for $E_\mathrm{L} < u < E_\mathrm{Th}$, given $I_\mathrm{syn} = 0$, the differential $\dot{u}(u)$ in Equation (2.14)



(a) LIF model (b) QIF model (c) EIF model

Figure 2.12: Sketch of one dimensional integrate-and-fire model bifurcation patterns. The three graphs show $\dot{u}(u)$ and the stationary points $\dot{u}(u) = 0$ for non-zero I_{syn}. Filled circles indicate stable stationary points, unfilled circles unstable stationary points. The reset mechanism is indicated by the unfilled box at $u = E_{Th}$ and the dashed arrow pointing back at u. The dotted grey lines show the same graph with a different choice for I_{syn}. Inspired by [Izh07].

evaluates to $\dot{u}(u) < 0$, even for infinitesimally small $E_{Th} - u$. As shown in Figure 2.12(a), it is only once u reaches E_{Th} that an output "spike" is issued and the neuron is reset.

This behaviour has two major shortcomings. The model does not produce a sharp spike-formed action potential with a sudden rise in the membrane potential, and the biological phenomenon of *spike latency* (Figure 2.11(i), [Izh04]) is not modelled: for a single short input current pulse the neuron might not spike immediately, but with a certain delay that decreases with the amplitude of the pulse.

The inner term in f merely shifts and rescales the membrane potential u, such that E_L maps to a dimensionless 0 and E_{Th} to 1 (before E_L is subtracted).

Spike generation and spike latency are modelled by non-linear integrate-and-fire neurons

$$-C_m \cdot \dot{u}(t) = g_L \cdot \left(f\left(\frac{u(t) - 2 \cdot E_L}{E_{Th} - E_L} \right) \cdot (E_{Th} - E_L) + E_L \right) + I_{syn}\,, \tag{2.17}$$

where f is some non-linear function. If f is chosen as the identity function, Equations (2.14) and (2.17) are equivalent.

Examples for non-linear integrate-and-fire models are the quadratic integrate-and-fire (QIF) and exponential integrate-and-fire (EIF) models: the corresponding functions f with free parameters f_1, f_2 are given as

$$f_{QIF}(u) = -f_1 \cdot (u - f_2)^2 \qquad f_{EIF}(u) = u - f_1 \cdot \exp(u - f_2)\,. \tag{2.18}$$

Care has to be taken when performing numerical integration of the equation, as the non-linearity might be numerically unstable.

As shown in Figures 2.12(b) and 2.12(c) the behaviour of these two models is qualitatively equivalent: as soon as u crosses a certain value E_{Th}^{eff}, the potential is quickly pushed towards E_{Th}. Analogously to the LIF model, for $u < E_{Th}^{eff}$ the membrane converges towards a stable stationary point. Location and existence of the two stationary points depends on the external input current I_{syn}.

2.3.6 *Two-dimensional Hodgkin-Huxley approximations: the AdEx model*

The above integrate-and-fire neuron models are one-dimensional: their state vector solely consists of the membrane potential u. As mentioned at the end of Section 2.3.4, one-dimensional models cannot account for many of the observed behavioural patterns in biological neurons, as the neuron state u is reset along with each output spike. Surprisingly, most of the relevant behaviour of the HH model (which has four state variables) can be modelled with only two-dimensions.

The Izhikevich model is a well-established and computationally cheap example of such an approximation [Izh04]. Another interesting approximation is the multi-timescale adaptive threshold (MAT) model, which minimally extends the LIF model with an adaptive threshold $E_{\mathrm{Th}}(t)$. Due to its linearity, the model is numerically stable and yet very successful in approximating membrane potential traces recorded from biological neurons [KTS09].

In this thesis we focus on the adaptive exponential integrate-and-fire (AdEx) model implemented in hardware in the HBP physical model system NM-PM1. Just as the above models, it is a two-dimensional non-linear integrate-and-fire model which reproduces a majority of the behavioural patterns observed in nature [BG05; GB09]. In addition to the membrane potential $u(t)$, the AdEx model tracks an adaptation current $I_{\mathrm{a}}(t)$, which hyperpolarises the neuron. The current $I_{\mathrm{a}}(t)$ increases by a small constant b with every generated output spike

The adaptation current $I_{\mathrm{a}}(t)$ can be biologically interpreted as a form of habituation or short-time plasticity [Kan+12].

$$I_{\mathrm{a}}(t) \leftarrow I_{\mathrm{a}}(t) + b \quad \text{on } u(t) > E_{\mathrm{Th}}\,. \tag{2.19}$$

The adaptation current I_{a} decays exponentially with a time constant τ_{a} and additionally depends on the "subthreshold adaptation conductance" $a \geq 0$ which controls the influence of the membrane potential on the decay rate. For $u(t) < E_{\mathrm{L}}$ the adaptation current decays faster, while $u(t) > E_{\mathrm{L}}$ prolongs the decay:

The subthreshold adaptation allows to model the biological phenomenon of "threshold variability", see Figure 2.11(o).

$$-\mathrm{d}/\mathrm{d}t\ \tau_{\mathrm{a}} \cdot I_{\mathrm{a}}(t) = I_{\mathrm{a}}(t) - a \cdot (u(t) - E_{\mathrm{L}})\,. \tag{2.20}$$

The model furthermore inherits the exponential spike generation mechanism from the EIF model as described in Equations (2.17) and (2.18). The final model equation is given as

In the original AdEx paper and corresponding code, $E_{\mathrm{Th}}^{\mathrm{exp}}$ is usually referred to as E_{Th}, and E_{Th} is replaced by a potential E_{spike}.

$$-C_{\mathrm{m}} \cdot \dot{u}(t) = g_{\mathrm{L}} \cdot (u(t) - E_{\mathrm{L}}) + I_{\mathrm{Th}}(u(t)) + I_{\mathrm{a}}(t) + I_{\mathrm{syn}}(u(t), t) \tag{2.21}$$

$$I_{\mathrm{Th}}(u) = g_{\mathrm{L}} \cdot \Delta_{\mathrm{Th}} \cdot \exp\left(\frac{u - E_{\mathrm{Th}}^{\mathrm{exp}}}{\Delta_{\mathrm{Th}}}\right)\,. \tag{2.22}$$

The exponential threshold potential $E_{\mathrm{Th}}^{\mathrm{exp}}$ controls the minimum membrane potential $u(t)$ at which the inner term of the exponential in $I_{\mathrm{Th}}(u)$ is positive. For larger membrane potentials an exponentially rising current triggers an avalanche which causes the generation of an output spike. The parameter Δ_{Th} controls the slope of the exponential

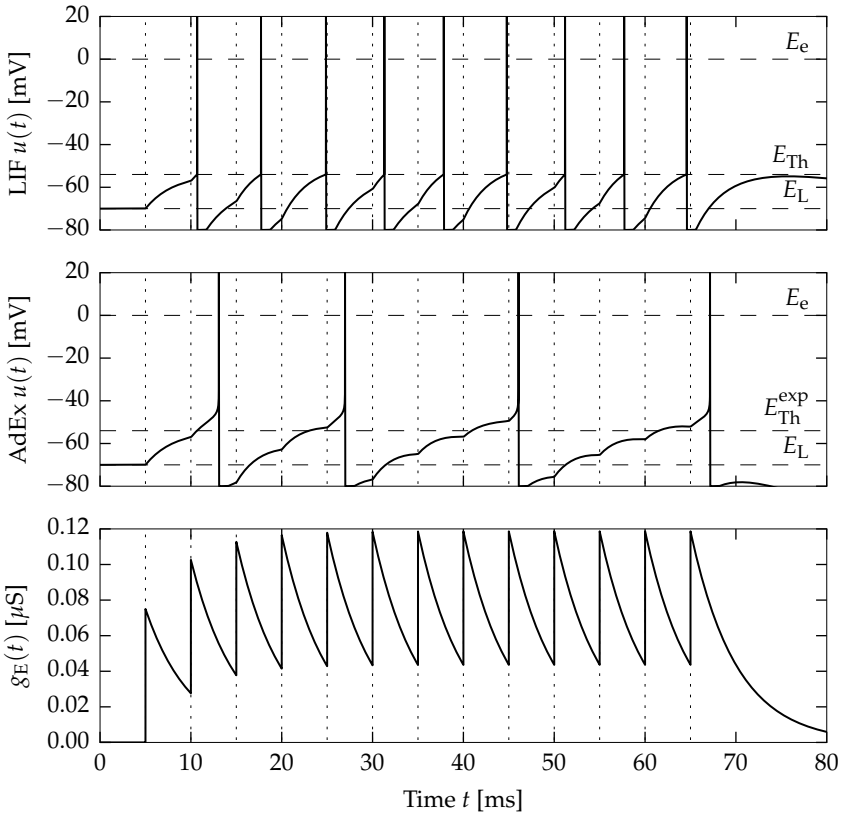


Figure 2.13: Comparison between the LIF and AdEx neuron models. The first two plots show the membrane potential of a LIF and an AdEx neuron, the bottom plot the conductance of a conductance based excitatory synapse, which receives 13 spikes in an interval of $\Delta t = 5\,\text{ms}$. Due to the adaptation current, the output spike rate of the AdEx neuron decreases over time, whereas the LIF neuron outputs spikes at a constant rate. The spike potential at $E_{\text{spike}} = 20\,\text{mV}$ is artificially inserted by the simulator for the LIF model, the AdEx model intrinsically generates a spike onset.

current. While the rising spike onset is explicitly modelled, the falling edge is not. The same reset mechanism as in the LIF model in Equation (2.16) is used, albeit the reset threshold E_{Th} is chosen considerably larger in the AdEx model.

Figure 2.13 depicts a comparison of the behaviour of a LIF and an AdEx neuron with a conductance based synapse for a series of input spikes with equidistant timing. As a result of the adaptation current the output spike rate reduces over time.

The AdEx model can emulate the simpler LIF model by setting the parameters a and b to zero and deactivating the exponential threshold current I_{Th}, which – depending on the implementation – can be achieved by setting Δ_{Th} to zero. Table 2.1 gives an overview of all parameters in the AdEx and LIF model with conductance based excitatory and inhibitory synapses.

		LIF AND ADEX MODEL PARAMETERS AND TYPICAL VALUES			
		Potentials			
		DESCRIPTION	LIF	ADEX	
	E_{L}	Membrane leak or resting potential	-65.0	-70.6	[mV]
	E_{Th}	Threshold potential. If passed, the neuron resets and issues an output spike.	-50.0	-40.0	[mV]
	E_{reset}	Reset potential. Potential the membrane is reset to during the refractory period.	-65.0	-70.6	[mV]
◇	E_{e}	Excitatory synapse reversal potential	0.0	0.0	[mV]
◇	E_{i}	Inhibitory synapse reversal potential	-70.0	-80.0	[mV]
		Time constants			
		DESCRIPTION	LIF	ADEX	
	τ_{ref}	Duration of the refractory state	0.1	0.1	[ms]
◇	τ_{e}	Excitatory synapse time constant	5.0	5.0	[ms]
◇	τ_{i}	Inhibitory synapse time constant	5.0	5.0	[ms]
		Membrane parameters			
		DESCRIPTION	LIF	ADEX	
	C_{m}	Membrane capacitance	1.0	0.281	[nF]
	g_{L}	Membrane leak conductance	0.05	0.03	[µS]
	τ_{m}	Membrane time constant ($\tau_{\mathrm{m}} = C_{\mathrm{m}}/g_{\mathrm{L}}$)	20.0	9.37	[ms]
		AdEx adaptation and exponential current mechanism			
		DESCRIPTION	LIF	ADEX	
○	a	Subthreshold adaptation	/	4.0	[nS]
○	b	Spike-triggered adaptation current	/	0.08	[nA]
○	τ_{a}	Adaptation current time constant	/	144.0	[ms]
○	$E_{\mathrm{Th}}^{\mathrm{exp}}$	Exponential threshold potential	/	-50.4	[mV]
○	Δ_{Th}	Exponential current slope	/	2.0	[mV]
		State variables			
		DESCRIPTION	LIF	ADEX	
	$u(t)$	Membrane potential	/	/	[V]
○	$I_{\mathrm{a}}(t)$	Adaptation current	/	/	[A]
◇	$g_{\mathrm{e}}(t)$	Excitatory channel conductance	/	/	[S]
◇	$g_{\mathrm{i}}(t)$	Inhibitory channel conductance	/	/	[S]

Table 2.1: Parameters of the LIF and AdEx neuron models and state variables in conjunction with excitatory and inhibitory conductance based synapses. The typical values reflect the biologically motivated default parameters in *PyNN* 0.8. ○ Only available in the AdExp model. ◇ Synapse parameters.

2.4 NEUROMORPHIC HARDWARE

Biological spiking neural networks, including the human brain, are asynchronous, distributed, extremely parallel and stochastic. Classical digital computers on the other hand are synchronous, centralised, of limited parallelism and deterministic. They are conceivably ill-suited for time and energy efficient simulation of large-scale spiking neural networks. The term *neuromorphic hardware* refers to systems which trade the versatility of general purpose computers with the architectural properties of central nervous systems and are specifically developed for a certain range of neural network models. Neural networks have been predominantly implemented in hardware in the 1950s and 1960s, when no powerful general purpose computers were available, see for example [HMW60; WH60]. However, no system for brain-scale networks has been developed to date.

The description of the hardware systems in this chapter follows their specification, comments regarding the current state of the systems at the time of writing are given in Chapter 5.

Providing such neuromorphic hardware platforms is a central aim of the HBP. Here, two complementary approaches are pursued. The physical model NM-PM1 simulates individual neurons and synapses as analogue physical model circuits. Conversely, the fully digital many-core system NM-MC1 consists of a vast number of conventional microprocessors, each of which simulates a small number of neurons. In both systems spikes are propagated over a digital, packet based, asynchronous and potentially unreliable communication network [Hum15c]. In this section we describe NM-MC1, NM-PM1, its single-chip predecessor "Spikey" and the software stack provided to the end-user.

2.4.1 *NM-MC1: The many-core system*

The neuromorphic many-core system NM-MC1 is developed at the University of Manchester and based on the SpiNNaker chip, a multiprocessor designed for real-time simulation of spiking neural networks [Hum15c]. Each chip contains up to 18 ARM968 processors running at a nominal frequency of 180 MHz, with one processor dedicated to management purposes. Each processor has access to 32 KiB of instruction memory and 64 KiB data memory, while each chip connects to 128 MiB of external DDR SDRAM. Additionally, SpiNNaker features six bidirectional inter-chip communication links with integrated router used to exchange spike events over the network of chips during simulation. NM-MC1 consists of boards with 48 SpiNNaker chips each, organised in a torus network topology. NM-MC1 will eventually consist of ten cabinets with 120 boards each, resulting in a maximum of 979 200 processors for neural network simulation [Pai+13; Fur+13].

The system is theoretically capable of running any neuron model. However, the processors do not feature a floating-point unit, so in order to avoid the overhead of a software floating-point implementation,

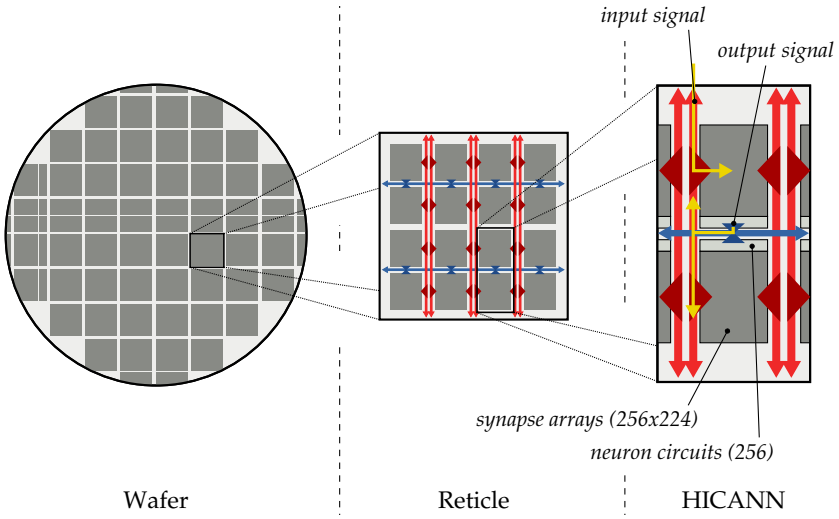


Figure 2.14: NM-PM1 wafer high level architecture overview. A single wafer in the NM-PM1 system consists of 384 HICANN chips, organised in reticles. A HICANN consists of two analogue blocks, each built of a synapse array and a neuron circuit. Each HICANN is connected to an on-chip network, organised in horizontal buses (blue) receiving neuron output, and vertical buses (red) transmitting input signals to the synapse drivers. Adapted from [Pet+14].

the neuron time dynamics are simulated with fixed-point arithmetic. As NM-MC1 is designed for the execution of spiking neural networks at biological timescale, the number of neurons per core varies with the computational complexity of the model. The system supports the LIF model with both conductance and current based synapses, as well as the Izhikevich neuron model [Hum15c; Izh04]. Algorithmically, the LIF neuron dynamics are integrated using the Euler method at 1 ms timestep with 32-bit intermediate fixed-point values and 16-bit fixed-point parameter storage, allowing up to 256 LIF neurons with conductance based synapses in a single core [Ras+10]. Correspondingly, NM-MC1 can simulate up to 250 million neurons, which is two orders of magnitude smaller than the estimated number of neurons in the human cortex [BS13].

2.4.2 *NM-PM1: The physical model*

The neuromorphic physical model NM-PM1 is developed at the Kirchhoff Institute for Physics at Heidelberg University. NM-PM1 is built around the mixed-signal HICANN (High Input Count Analogue Neural Network) chip. Each HICANN consists of two blocks, each hosting 256 analogue neuron circuits and a 224×256 matrix of analogue synapse circuits. Adjacent rows in the synapse matrix receive external input via synapse drivers (112 per block), which are connected to

Research on HICANN and NM-PM1 originated in the European FACETS and BrainScaleS projects and is now continued in the HBP.

a digital on-chip communication network. To provide neurons with variable synapse count, up to 64 physical neuron circuits can be joined to form a logical neuron, allowing up to 14 336 synapses per logical neuron at the cost of reducing the number of logical neurons per block to a minimum of four. Each synapse can store a 4-bit weight. Synapse rows can be combined in order to achieve a higher weight resolution.

The analogue circuits on the chip emulate the dynamics of the AdEx and LIF models with excitatory and inhibitory conductance based synapses. Due to the analogue implementation it is important to distinguish model and hardware parameters: individual hardware parameters are mostly configured as voltages in analogue floating gates. The biological model parameters must be mapped onto these hardware parameters, taking calibration values for the individual circuits into account. As each neuron circuit can only be configured to use one of two membrane capacitances, the mapping process must not only adapt the model membrane potentials, currents, and conductivities to their hardware voltage representation, but also rescale the time constants to match the model membrane capacitance C_m. This results in a speedup factor between 10^3 and 10^5 compared to biological timescale.

Alternatively, the speedup factor can specified by the user, but then only limited choices regarding the membrane capacitance C_m are possible.

Apart from the 10^5 speedup factor, another salient property of NM-PM1 is its wafer-scale integration: instead of separating the individual HICANN chips from their silicon wafer after manufacturing, the wafer is left intact. Connections between the largest lithographic units – the reticles – are layered onto the wafer in a post-processing step. The wafer-scale approach is feasible, as errors in the analogue circuitry are tolerable (as they are in biological systems) and can be marked as such in software. Each wafer comprises 384 HICANNs, summing up to a theoretical maximum of 1 966 080 neurons and 44 million synapses [Pet+14]. An overview of the organisational topology is depicted in Figure 2.14. In its final stage, NM-PM1 is planned to consist of 20 wafer systems, resulting in a total of 3.9 million neurons [Hum15c]. The executable system specification (ESS) allows to simulate parts of NM-PM1 without access to the actual hardware [Brü+11].

While impressive, the number of neurons in NM-PM1 is still four magnitudes smaller than the estimated number of neurons in the human brain, but already close to the size of a mouse cortex [BS13].

2.4.3 *Spikey*

The Spikey analogue neuromorphic hardware system (Figure 1.1) has been developed as part of the FACETS project at the University of Heidelberg. As a predecessor to HICANN, the single Spikey chip in the system offers a speedup factor of 10 000 compared to biological timescale, and 384 analogue neurons, split into two blocks of 192 neurons each. Each neuron implements a limited LIF model and connects to 256 configurable analogue synapses with 4-bit weight resolution. Synapses are organised in 256 lines per block, with each line passing inputs from external or internal sources to the synapses. The neuron parameters τ_{ref} and g_L can be chosen per neuron, all other neuron parameters are shared by groups of 96 neurons [Pfe+13].

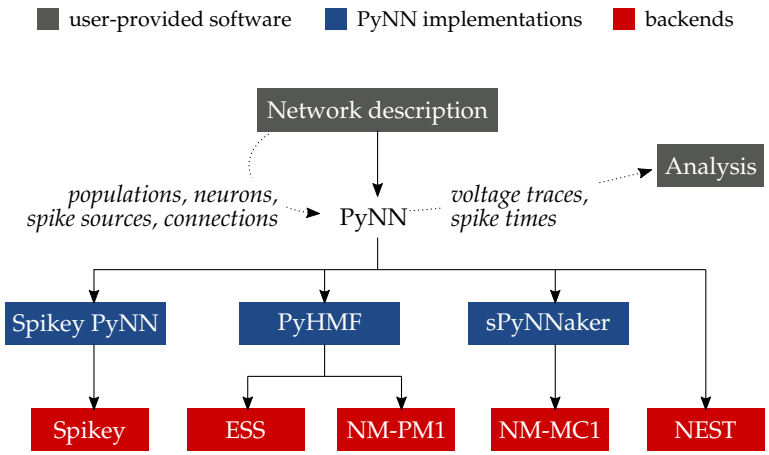


Figure 2.15: Neuromorphic hardware system software stack and data flow. Users provide a network description to *PyNN* and request the recording of certain variables. Implementations of the *PyNN* API (blue) then communicate with the backend specific software (red). An interface for NEST is directly included in *PyNN*.

2.4.4 *Software stack*

Usually, neuromorphic hardware platforms, their emulators, and software simulators come with a native software interface specifically tailored to the system. For neuromorphic hardware, the backends map the network graph onto the neuromorphic substrate (place-and-route), convert the abstract neuron model parameters to concrete hardware parameters and perform the necessary communication tasks.

While the platform-provided libraries allow users to exploit specific platform features, their variety hinders the development of cross-platform network simulations, as each platform has to be targeted individually. To overcome this limitation, an API with the name *PyNN* is developed as part of HBP [Dav+08]. As shown in Figure 2.15, *PyNN* specifies a common software interface that allows to construct spiking neural network graphs, inject spike sources and flag neuron spike times and membrane potentials for recording. The individual developers of the hardware or software simulators provide an implementation of the *PyNN* interface that communicates with the corresponding backend. This allows code written on top the *PyNN* framework to run on arbitrary platforms, as long as it provides the required neuron models and the parameters are in the supported range.

Platforms targeted in this thesis via *PyNN* are Spikey, NM-PM1 and its emulation ESS, NM-MC1 and the software simulator NEST. NEST is developed at the Forschungszentrum Jülich as part of the research on large scale simulation of brain models on conventional high performance computing platforms in the HBP [GD07]. By design, NEST is the most versatile and mathematically exact of the targeted platforms and acts as a reference system in this thesis.

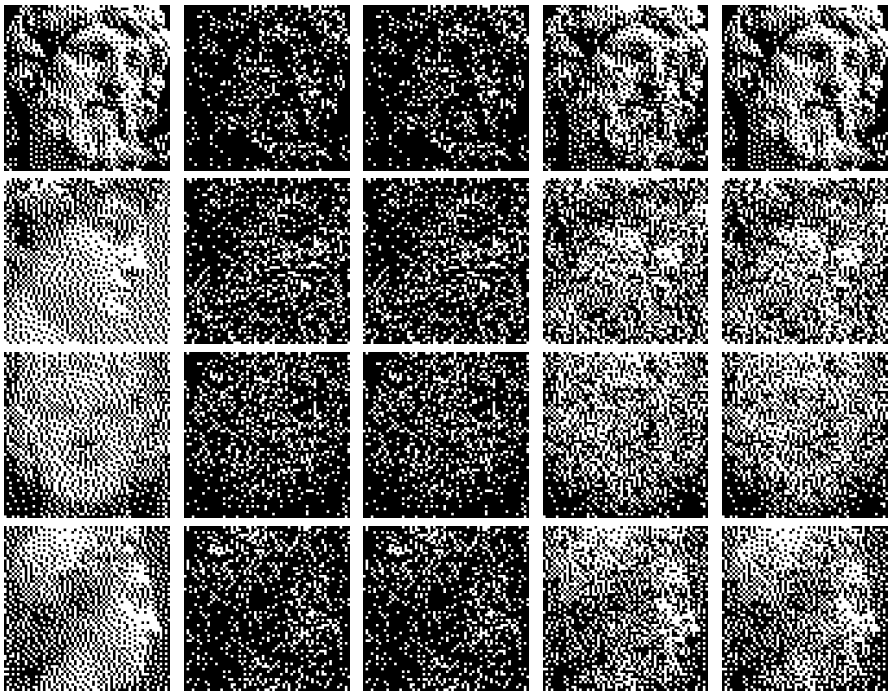


Figure 2.16: Example of pattern completion with a Hopfield associative memory. Column (a) shows 64×64 1-bit images encoded as 4096-dimensional column vectors $\vec{x}_k$. When presenting random 30% of the original image as clue $\vec{x}$ to the memory (column (b)), it iteratively completes the patterns ((c)-(e)). Interference with four other stored images (not shown here) causes imperfect reproduction of the originals. The experiment is repeated with a BiNAM in Figure 2.20 (where all patterns are shown).

2.5 THE WILLSHAW ASSOCIATIVE MEMORY MODEL (BINAM)

Along with artificial neural networks, technical implementations of associative memories have been researched since the middle of the last century, whereas the study of "associations" itself dates back to the ancient Greece philosopher Aristotle [War16]. From our intuition it seems to be obvious that associations are an integral part of cognition: our brain constantly associates sensory input with internal states, such as feelings and memories, even if the two associated items are only connected remotely: consider the association of the smell of dry wood with the feeling of warmth at a fireplace. Of course, associations do not solely occur as a response to sensory input. Instead, they also play an important role in internal reasoning: often our mind follows a sequential chain of associations from one thought to the next until it suddenly "snaps" to the missing piece we have been searching for [Pal13].

Artificial associative memories aim at being high-level abstractions of the above concepts and merely touch the question on how associations actually work in human cognition. On the other hand, many

associative memory models are implementable as neural networks and could be useful building blocks in artificial brain models. This section gives a quick conceptual overview of artificial associative memories and continues with a thorough description of the Willshaw model, its properties and implementation as a first-generation neural network.

2.5.1 *Artificial associative memory models*

Artificial associative memory models usually feature two phases: a *training phase* in which input vectors $\vec{x}_k$ and the corresponding association $\vec{y}_k$ are presented to the model. In the *recall phase* an arbitrary input vector $\vec{x}$ is fed into the system. In the optimal case the memory responds with the previously trained $\vec{y}_k$ corresponding to the trained input $\vec{x}_k$ to which $\vec{x}$ is closest according to a dissimilarity measure $\eth$

$$f(\vec{x}) = \vec{y}_k \quad \text{where} \quad k = \arg\min_k \eth(\vec{x}_k, \vec{x})\,. \tag{2.23}$$

This concept resembles content addressed memory: data is not accessed by a physical address but by data itself, comparable to a hash map in computer science [Koh12]. Associative memories should also be clearly distinguished from function approximation in machine learning: the goal of associative memories is not to learn a generalised, continuous mapping between $\vec{x}$ and $\vec{y}$, but to respond with one of the explicitly trained output vectors $\vec{y}_k$.

We distinguish two operational modes for associative memories: *auto-association*, in which $\vec{x}_k = \vec{y}_k$ for all samples k, and *hetero-association*, for which this equality is not presumed. As shown in Figure 2.16, auto-association can be interpreted as pattern-completion: given an altered (noisy) clue $\vec{x}$ of a previously trained vector $\vec{x}_k$, the output of the memory converges towards a state resembling the original $\vec{x}_k$. Conversely, hetero-associations can be interpreted as semantic links between input $\vec{x}_k$ and output $\vec{y}_k$ [Pal13].

Hopfield networks, proposed in 1982, are one of the most famous associative memory models: they consist of a fully-connected network of McCulloch-Pitts cells (Section 2.1.1) and operate on binary input and output vectors. During training, the synaptic weights w_{ij} are set according to the *Hebbian* learning rule [Heb05]. A connection between neuron i and j with $i \neq j$ is set to $w_{ij} = 1$ if $(\vec{x}_k)_i$ positively correlates with $(\vec{y}_k)_j$. Otherwise w_{ij} is set to $w_{ij} = -1$

$$w_{ij} = \operatorname{sgn}\left(\sum_k \left((\vec{x}_k)_i - \tfrac{1}{2}\right) \cdot \left((\vec{y}_k)_j - \tfrac{1}{2}\right)\right)\,. \tag{2.24}$$

With this correlation based training scheme, the recurrent, fully-connected network acts as a dynamical system with the trained output vector imprinted as attractors [Hop82; Hop07]. Given an initial state $\vec{x}$, the network is likely to converge to the trained output $\vec{y}_k$ (Figure 2.16).

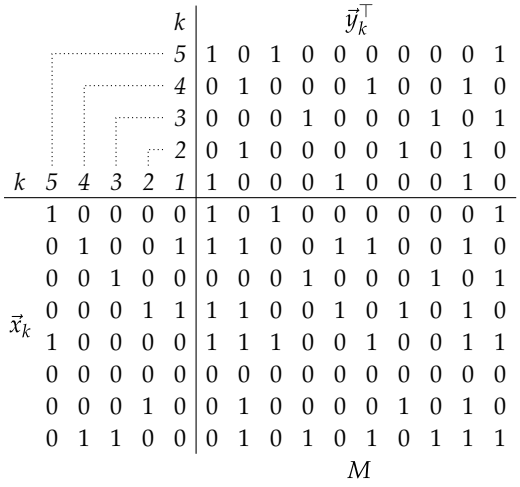


Figure 2.17: Example of a 8×10 storage matrix M after five samples $(\vec{x}_k, \vec{y}_k)$ with $c = 2$ ones in each input vector and $d = 3$ ones in each output vector have been trained according to Equation (2.27). The dotted lines connect input and output pairs.

2.5.2 *Formal description of the Willshaw model*

The output of the memory can of course be fed back to its input, which would result in a dynamical system and cause lots of fascinating behaviour. This is out of scope for this thesis.

In contrast to Hopfield networks, the basic Willshaw associative memory model, or Binary Neural Associative Memory (BiNAM), does not describe a dynamical system. The model can be traced back to a paper by Steinbuch in 1961 [Ste61] and was independently described in 1969 by Willshaw et. al. as a parallel, non-local and fault-tolerant associative network [WBL69]. It was further formalised and analysed by Palm [Pal80]. Extensions of the model – especially for spiking networks – have, amongst others, been proposed by Knoblauch [Kno03; Kno+14]. Yet, as expounded in the introduction, we stick to the basic model.

Mathematically a BiNAM can be defined as a binary storage matrix $M \in \mathbb{B}^{m \times n}$, where $\mathbb{B} = \{0, 1\}$ is the base set of Boolean algebra, m is the dimensionality of the binary input vector $\vec{x} \in \mathbb{B}^m$ and n is the dimensionality of the output vector $\vec{y} \in \mathbb{B}^n$.

Training When training an association between an input $\vec{x}_k$ and output $\vec{y}_k$, the storage matrix M is updated to a new M'

$$M' = M \vee \left(\vec{x}_k \cdot \vec{y}_k^\top \right) , \tag{2.25}$$

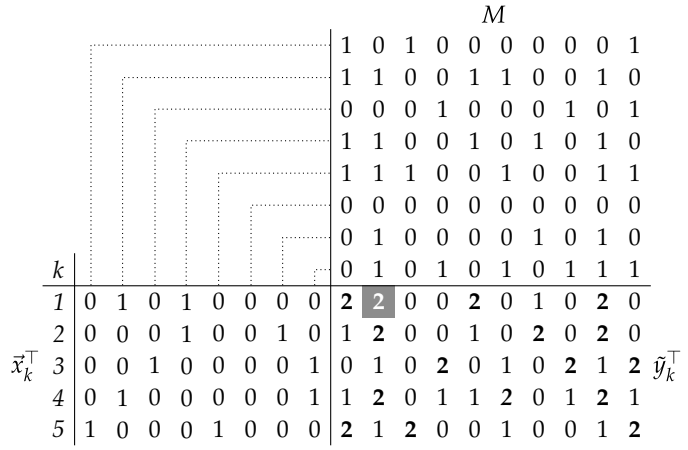
The condition $\vec{x}_k = \vec{x}_\ell \Leftrightarrow k = \ell$ ensures unique input vectors: while it is possible for multiple $\vec{x}$ to map to the same $\vec{y}$, it is prohibited for identical $\vec{x}$ to map to multiple $\vec{y}$.

where "$\vee$" is the element-wise "OR"-operation from standard Boolean algebra. If a set of N samples $(\vec{x}_k, \vec{y}_k)$

$$\mathfrak{D} = \{(\vec{x}_k, \vec{y}_k) \mid k \in \{1, \ldots, N\}\} \text{ with } \vec{x}_k = \vec{x}_\ell \Leftrightarrow k = \ell \tag{2.26}$$

is given in advance, a pre-calculated storage matrix M can be obtained according to the following expression (Figure 2.17)

$$M = \bigvee_{j=1}^{N} \vec{x}_j \cdot \vec{y}_j^\top . \tag{2.27}$$



									1	0	1	0	0	0	0	0	1
									1	1	0	0	1	1	0	0	1	0
									0	0	0	1	0	0	0	1	0	1
									1	1	0	0	1	0	1	0	1	0
									1	1	1	0	0	1	0	0	1	1
									0	0	0	0	0	0	0	0	0	0
									0	1	0	0	0	0	1	0	1	0
k									0	1	0	1	0	1	0	1	1	1
1	0	1	0	1	0	0	0	0	**2**	2	0	0	**2**	0	1	0	**2**	0
2	0	0	0	1	0	0	1	0	1	**2**	0	0	1	0	**2**	0	**2**	0
3	0	0	1	0	0	0	0	1	0	1	0	**2**	0	1	0	**2**	1	**2**
4	0	1	0	0	0	0	0	1	1	**2**	0	1	1	**2**	0	1	**2**	1
5	1	0	0	0	1	0	0	0	**2**	1	**2**	0	0	1	0	0	1	**2**

Figure 2.18: Recalling the values associated to the $\vec{x}_k$ previously trained in Figure 2.17. The lower-right quadrant shows the intermediate results $\tilde{y}_k^\top = \vec{x}_k^\top \cdot M$ before applying the threshold function Θ_c. Bold numbers correspond to those values that would be set to one in the final output, the grey backdrop signals a false-positive. The dotted lines connect rows in M with their corresponding factors in the input data used when performing the vector-matrix multiplication.

In this thesis we assume that such a pre-calculated storage matrix M is already available – we do not try to implement online training of the network. Additionally, and though not required for the operation of the BiNAM, analysis of the memory is simplified significantly, if the number of bits set to "one" in both input and output vector is constant. We denote the number of "ones" in the input vectors $\|\vec{x}_k\|_1$ as c and the number of "ones" in the output vectors $\|\vec{y}_k\|_1$ as d.

Generation of data $\mathfrak{D}$ as used in the experiments conducted in this thesis is discussed in detail in Section 3.3.

Recall In the recall phase, an arbitrary input vector $\vec{x}$ is given. The output of the memory $\vec{y}$ can then be calculated as follows

$$\vec{y} = \Theta_c(\tilde{y}) = \Theta_c\left((\vec{x}^\top \cdot M)^\top\right) \quad \text{with} \quad c = \|\vec{x}\|_1 = \sum_{i=0}^{m} (\vec{x})_i = \vec{x}^\top \cdot \vec{x}\,, \tag{2.28}$$

where $\Theta_\theta : \mathbb{N}^m \longrightarrow \mathbb{B}^m$ is a step function with threshold θ, which maps the intermediate integer results of the matrix-vector multiplication $\tilde{y}$ to binary values

$$(\Theta_\theta(\vec{z}))_i = H\Big((\vec{z})_i - \theta\Big)\,. \tag{2.29}$$

Although the equation for Θ_θ looks fairly innocent, its realisation is the most crucial requirement for an operational spiking BiNAM implementation.

Figure 2.18 shows the recall rule applied to the storage matrix M previously trained in Figure 2.17. As demonstrated, the recalled output of the BiNAM is not perfect. A *false positive* has been introduced (highlighted in grey): a one is present in component j of the recalled output for $\vec{x}_k$, although the trained $(\vec{y}_k)_j$ was set to zero.

2.5.3 *Choice of the threshold θ*

Until now, the choice of $\theta = c$ in Equation (2.28) has not been motivated. Consider a BiNAM trained for a single sample $(\vec{x}_k, \vec{y}_k)$. The memory matrix is now set to $M = \vec{x}_k \cdot \vec{y}_k^\top$. Trying to recall $\vec{y}_k$ by placing $\vec{x}_k$ in Equation (2.28) yields

$$\tilde{y} = (\vec{x}_k \cdot \vec{y}_k^\top)^\top \cdot \vec{x}_k = \vec{y}_k \cdot (\vec{x}_k^\top \cdot \vec{x}_k) = \vec{y}_k \cdot c\,. \tag{2.30}$$

Consequently, if a single sample is stored in the memory, the network returns an exact copy of the output vector $\vec{y}$ scaled by c. Due to the additive superposition caused by the "$\vee$" in the training phase, newly trained samples can never lower the value of a single component i in $\tilde{y}$, but only cause an increase towards the upper limit c

$$(\tilde{y})_i = (M^\top \cdot \vec{x}_k)_i \leq (M'^\top \cdot \vec{x}_k)_i \leq (\mathbb{1} \cdot \vec{x}_k)_i \leq c\,, \tag{2.31}$$

where $\mathbb{1}$ is the matrix of "ones" and

$$M' = M \vee (\vec{x} \cdot \vec{y}^\top) \text{ for any } \vec{x} \in \mathbb{B}^m, \vec{y} \in \mathbb{B}^n\,. \tag{2.32}$$

Given these considerations, adaptively setting the threshold θ to the maximum possible value $c = \|\vec{x}_k\|_1$, is the most sensible solution, as it minimises the chance of a false positive – a bit in $\vec{y}$ being set to one although it should be zero – and yet prevents any *false negatives*: all trained ones in the output are always present (see also Section 2.5.6).

2.5.4 *Storage capacity and sparsity*

One of the defining properties of any memory is the amount of information that can be stored in the system. We refer to this measure as *storage capacity*, or – according to information theory – *information* or *entropy*. Let us first consider the case of a conventional binary memory matrix M of the size $m \times n$. Given a row index $k \in \{1, \ldots, m\}$ we can access any stored output vector $\vec{y}_k$. Each cell in $\vec{y}_k$ has two possible states, so the total number of possible $\vec{y}_k$ is 2^n, and the number of possible matrices M is $(2^n)^m$. According to information theory, the number of bits I needed to represent that many states is [Sha48]

$$I = \mathrm{lb}((2^n)^m) = m \cdot \mathrm{lb}(2^n) = m \cdot n\,, \tag{2.33}$$

where lb is the binary logarithm. Given our constraint $\|\vec{y}_k\|_1 = d$, the amount of information is given as

$$I = \mathrm{lb}\left(\binom{n}{d}^m\right) = m \cdot \mathrm{lb}\binom{n}{d}\,. \tag{2.34}$$

For auto-associative storage measures see [Pal80].

The same idea for the calculation of I can be applied to associative memories in the hetero-association mode [Pal80]. The only important

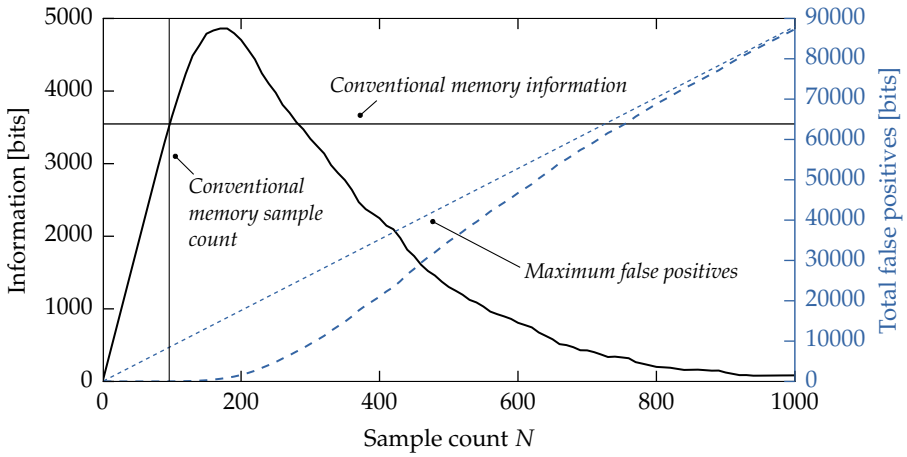


Figure 2.19: BiNAM information and false positive count over number of trained samples. Data size is $m = n = 96$, with $c = d = 8$. The optimal sample count is $N = 172$ with $I = 4859$. For a conventional memory the same measures are $N = 96$ and $I = 3\,547$.

difference between conventional and associative memories is the indexing scheme: output vectors $\vec{y}_k$ are not accessed by a row index k but by a unique vector $\vec{x}_k$. In the case that all N trained samples can be recalled perfectly, the storage capacity would be:

$$I = N \cdot \mathrm{lb} \binom{n}{d} \tag{2.35}$$

However, as explained in Section 2.5.3, the probability of false positive bits in the output increases with the number of trained samples. Given the number of false positives n^k_{fp} for a specific sample k, the possible number of states occupied by those wrong bits needs to be subtracted in the information calculation:

$$I = \sum_{k=1}^{N} \mathrm{lb} \binom{n}{d} - \mathrm{lb} \binom{d + n^k_{\mathrm{fp}}}{d} \tag{2.36}$$

The binomial coefficient

$$\binom{d + n^k_{\mathrm{fp}}}{d}$$

expresses the number of ways in which the correct d ones can be distributed amongst the total number of ones in the output.

If n^k_{fn} false negatives are present in the output for sample k (Section 2.5.6), an adapted equation must be used [RS91]:

$$I = \sum_{k=1}^{N} \mathrm{lb} \binom{n}{d} - \mathrm{lb} \binom{n^k_{\mathrm{fp}} + d - n^k_{\mathrm{fn}}}{d - n^k_{\mathrm{fn}}} - \mathrm{lb} \binom{n - n^k_{\mathrm{fp}} - d + n^k_{\mathrm{fn}}}{n^k_{\mathrm{fn}}} \tag{2.37}$$

As false positive and negative counts n^k_{fp} and n^k_{fn} are specific to the stored dataset and the actual $\vec{x}$ used for the recall operations, the storage capacity depends – in contrast to conventional memory – on memory content. Figure 2.19 shows a simple experiment: a BiNAM is trained with random data one sample at a time. After each sample has been trained, the stored information is calculated according to Equation (2.36), resulting in a characteristic information over sample

Achieving a higher storage capacity than conventional memories can be interpreted as intrinsic data compression in the BiNAM. Given the information in the input $\vec{x}$, the $\vec{y}_k$ can be decompressed.

count curve: the storage capacity I quickly increases with the number of stored samples N, up to a certain maximum, and then converges to $I = 0$ for large sample counts. In this particular example both the storage capacity I and the number of stored samples significantly outperform the conventional memory.

However, this comes at a cost: for once, there is a chance of false positives in the output (the memory is not perfect) and the high storage capacity can only be reached if input and output vectors are *sparse*: optimal performance is achieved if the number of ones c, d are chosen logarithmic with respect to m, n. For non-sparse data the memory matrix M converges to $\mathbb{1}$ too quickly, causing an increased false positive count n_{fp}^k.

Note that the above restriction does not imply that the BiNAM will not operate properly for non-sparse data – the maximum storage capacity and the corresponding optimal sample count will just be smaller than that of conventional memories of the same size. An example of dense data storage in a BiNAM is shown in Figure 2.20, which repeats the introductory pattern completion experiment from the beginning of this chapter with a BiNAM instead of a Hopfield network. In contrast to the Hopfield network, the BiNAM recalls all stored images perfectly in a single step.

Another conceptual property of the BiNAM model is that it can be interpreted as a generalisation of a conventional memory. It gracefully reduces to such a memory if the $\vec{x}$ are appointed to "column selectors" which contain a single "one", corresponding to $c = \|\vec{x}\| = 1$. Let, without any loss of generality, assume that the $\vec{x}_k$ are sorted such that $(\vec{x}_k)_k = 1$ and $N = m$

$$M = \bigvee_{k=1}^{m} \vec{x}_k \cdot \vec{y}_k^\top = (\vec{y}_1, \dots \vec{y}_m)^\top \Rightarrow M^\top \cdot \vec{x}_k = \vec{y}_k \,. \tag{2.38}$$

2.5.5 *Neural network implementation*

Since its inception, the BiNAM model has been proposed as "biologically plausible": it can be easily implemented as a first- or second-generation artificial neural network, and the corresponding topology resembles certain structures in the brain [WBL69; Pal80]. Yet, before we consider the implementation, two notes should be taken into account: as already mentioned, our implementation is not capable of online training. Furthermore, it does not provide dynamic threshold adaptation. The impact of a fixed threshold θ is discussed in Section 2.5.6.

Of course, the model can also be implemented as firing-rate network: in this case, the threshold function Θ_θ has to be used as non-linearity f.

Due to its binary nature, the BiNAM model can be perfectly implemented with the McCulloch-Pitts neuron model (Section 2.1.1). The network as shown in Figure 2.21 consists of a single layer of n neurons which receive the input signal $\vec{x}$, either as external input or from a sub-network. Each neuron corresponds to a single output component

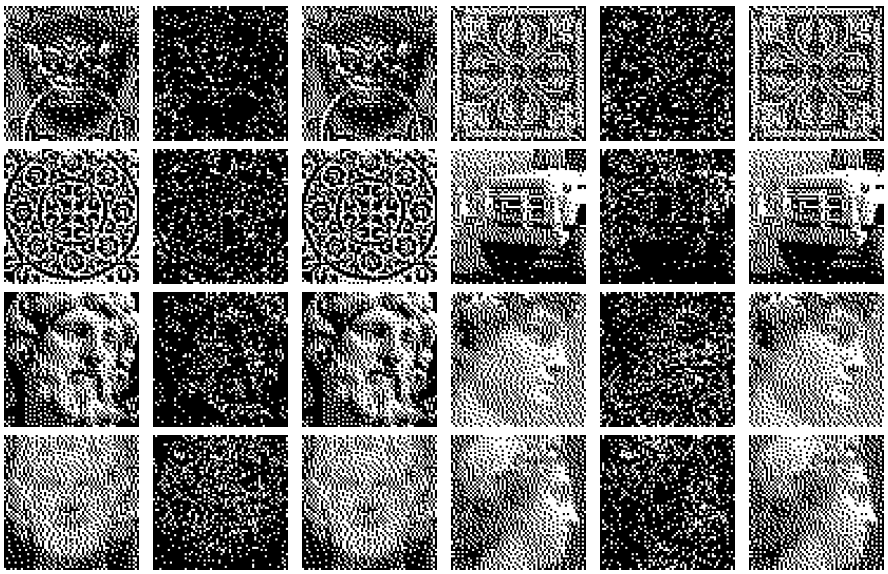


Figure 2.20: BiNAM pattern completion experiment with non-sparse vectors: the binary 64×64 pixel images in the (a) columns are trained as 4096-dimensional vectors $\vec{x}_k$. Random 30% of the "ones" in the original image are then presented as clue $\vec{x}$ to the memory (b). The recalled result is shown in (c). In this example all images are perfectly recalled by the BiNAM.

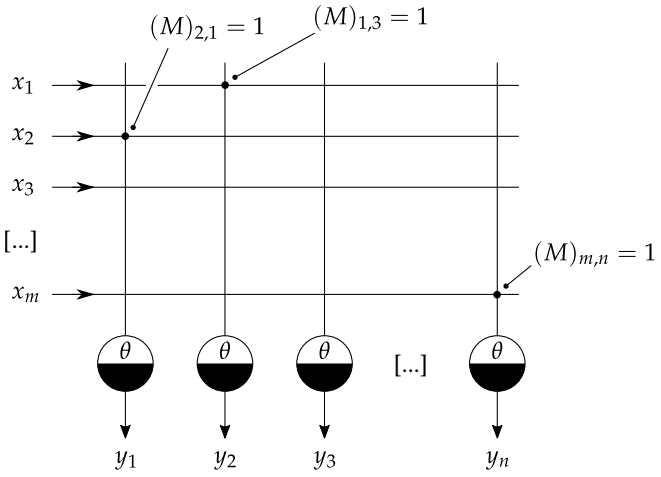


Figure 2.21: Neural network implementation of the BiNAM with McCulloch-Pitts cells. The $x_1, \ldots, x_m$ correspond to the components of an input vector $\vec{x}$, the $y_1, \ldots, y_n$ to the components of the output vector $\vec{y}$. Dots in the intersections between the neural input and the input components correspond to an excitatory synapse. These are inserted for all i, j with $(M)_{i,j} = 1$. The threshold θ is fixed in this implementation.

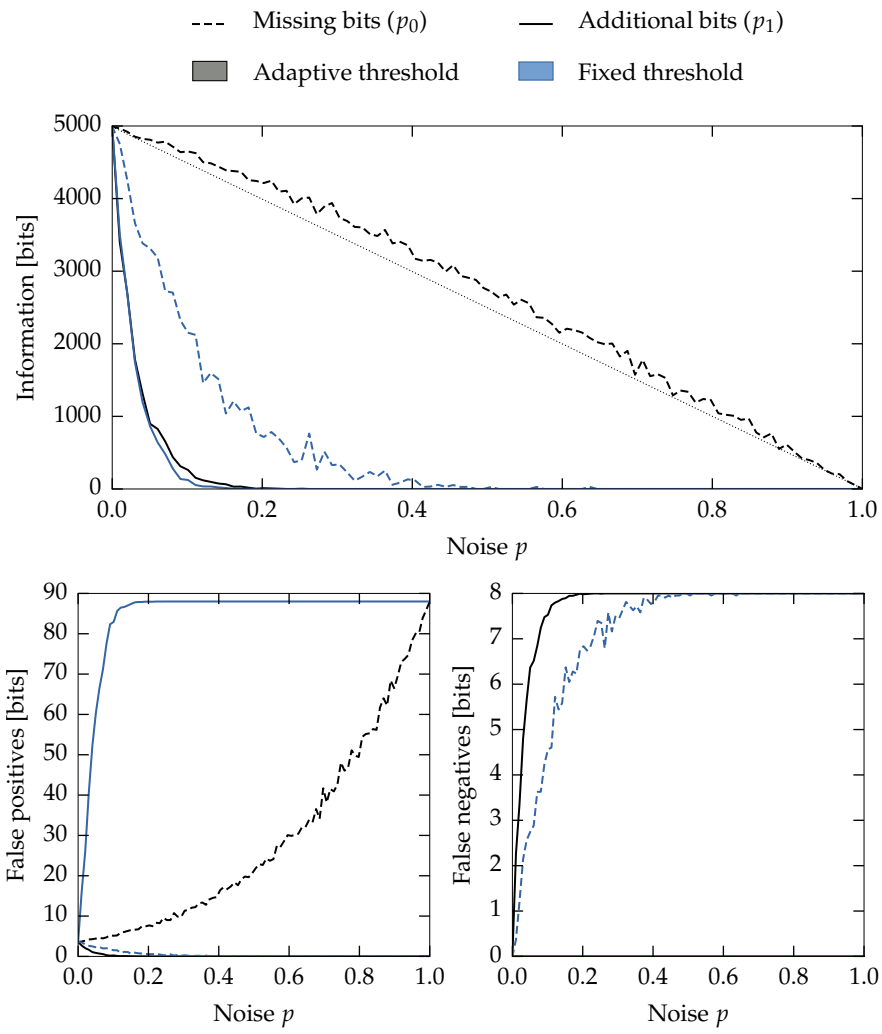


Figure 2.22: Information measure and error count over noise parameters p_0 and p_1. A BiNAM of size $m = n = 96$ is trained with $N = 172$ samples with $c = d = 8$ one-bits. The graphs show the total information and the averaged false positive and false negative counts for the recall of all trained samples. Addition and omission of bits are performed with probabilities p_0 and p_1. The behaviour between fixed and adaptive threshold θ is compared. In the fixed threshold case $\theta = m = 8$.

in $\vec{y}$. An excitatory synapse is added between input component i and output neuron j if, and only if, $(M)_{i,j} = 1$. The threshold θ should be set to the average number of "ones" c in the trained input. Smaller or larger values than c may be required if noise is present in $\vec{x}$.

2.5.6 *Impact of noise*

Aside from the pattern completion example in Figure 2.20, perfect conditions were assumed up to this point: the input vectors $\vec{x}_k$ presented to the network in the recall phase are the same as in the input data $\mathfrak{D}$ actually learned in the training phase. As noted above, even in this scenario there is the possibility of *false positives*. This failure mode results

from the storage matrix continuously getting saturated during training. If the "ones" in the training samples are uniformly distributed, the memory matrix M asymptotically approaches the "ones" matrix $\mathbb{1}$ (Section 3.3.2).

There are two ways in which we can corrupt the originally trained input vectors $\vec{x}_k$: by addition (additional bits are set to one) and by omission (bits originally set to one are set to zero). In the case of additive noise, the input vector $\vec{x}$ can be modelled as

$$(\vec{x})_i = (\vec{x}_k)_i \vee \eta \text{ with } \Pr(\eta = 0) = 1 - p_1 \text{ and } \Pr(\eta = 1) = p_1\,, \quad (2.39)$$

where p_1 is the probability of an arbitrary input vector component being set to one. In the omission (multiplicative) case, the input vector $\vec{x}$ can be modelled as

$$(\vec{x})_i = (\vec{x}_k)_i \wedge \eta \text{ with } \Pr(\eta = 0) = p_0 \text{ and } \Pr(\eta = 1) = 1 - p_0\,, \quad (2.40)$$

where p_0 is the probability of an arbitrary input vector component being set to zero.

Figure 2.22 shows the impact of the two kinds of noise on storage capacity, *false positive* and *false negative* counts. It furthermore distinguishes between an adaptive threshold $\theta = \|\vec{x}\|_1$ and a fixed threshold θ. For multiplicative noise and adaptive threshold, the information I decreases almost linearly with p_0. Counter-intuitively, the false-positive count increases the more bits are omitted in the input. This is caused by the missing input bits lowering the threshold θ. For a fixed threshold, the false-positive count converges to zero with increasing p_0, accompanied with an increase in the false-negative count. Correspondingly, the information measure drops to zero.

Additive noise has a much stronger negative impact on the storage capacity as its multiplicative counterpart. First of all, this is caused by p_1 and p_0 not being directly comparable: due to sparsity, the probability of adding an additional "one" to the input is higher than setting an actual "one" to zero in the multiplicative case. If no adaptive threshold is used, the false positive count quickly rises towards the maximum. With adaptive threshold, the addition of input bits causes false negatives as the increased threshold masks the correct "ones". In both cases the information measure I drops to zero.

To summarise the possible BiNAM failure modes: false positives can always be produced, even without noise. False negatives occur if input bits are missing and the threshold is not adapted (as in our neural network implementation, Section 2.5.5), and if θ is increased above c due to additional input bits.

2.6 SUMMARY AND OUTLOOK

This chapter provided a scenic overview of various fields of research. We discussed neural network models, both from the view of neurobiology and neuroinformatics, and their implementation in neuromorphic

hardware. Furthermore, we have thoroughly investigated the Willshaw associative memory and provided an implementation in the form of an artificial, first-generation neural network.

In Chapter 3, we combine these building-blocks: we extend our BiNAM implementation towards spiking neural networks and provide the workflow for design space exploration. In Chapter 4, we dive back into the details of the LIF and AdEx spiking neural network models to find proper values for the neuron parameters listed in Table 2.1. Finally, in Chapter 5, we pioneer into the realm of neuromorphic hardware and boldly execute the model on platforms, on which no associative memory has run before.

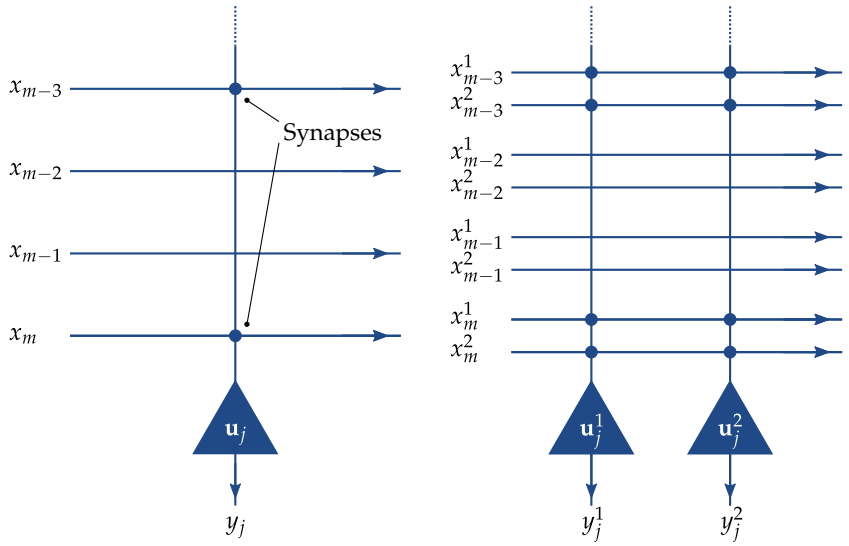
3 SPIKING ASSOCIATIVE MEMORY ARCHITECTURE AND TESTING

The previous chapter introduced the Willshaw associative memory model (BiNAM) and the notion of spiking neural networks. In this chapter we merge those two concepts: Section 3.1 describes the transition of the BiNAM architecture from a firing-rate coded to a spiking neural network. Furthermore, to account for the spiking network time dynamics, we specify the general testing procedure, how input and output data are encoded as a sequence of spikes, and how noise in the input data is parametrised in a spiking environment. Section 3.2 introduces multiple measures for the quantification of the network performance, while Section 3.3 focuses on the generation of datasets that can be used for network evaluation.

The spiking implementation of the Willshaw associative memory is no longer truly "binary" (regarding weights and timings). Therefore the name "BiNAM" might be a little misleading when referring to spiking neural networks; however, the same name is used in this thesis for the sake of consistency.

3.1 NEURAL NETWORK TOPOLOGY AND DATA ENCODING

The most trivial way of implementing the Willshaw associative memory model as spiking neural network is to directly employ the firing-rate network topology introduced in Section 2.5.5: if an entry in the trained storage matrix $(M)_{ij}$ is set to "one", a synaptic connection between



(a) Single neuron (b) Population of $K = 2$

Figure 3.1: Two possible implementations of a j-th BiNAM column as spiking neural network: in (a) the network is implemented as a one-to-one translation of the firing-rate neural network topology. In (b) each neuron is replaced by a population of K neurons ($K = 2$ in this example); memory input and output components are represented by K independent signals, $y^1_j, \ldots, y^K_j$.

the neuron j in the output layer and input i in the input layer is established. This basic topology is sketched in Figure 3.1(a). Another possible topology is shown in Figure 3.1(b): instead of single neurons, a population of neurons represents each output component, with input and output being transmitted via connection bundles. This topology is presented in detail in Section 3.1.3.

Difficulties in building a spiking neural network arise from the time dynamics: we need to think about how input and output data are represented as a sequence of spikes in the time domain. We consider this problem in more detail in Sections 3.1.1 and 3.1.2.

Furthermore, synapse weights are no longer dimensionless scalars which can be trivially mapped to the ones and zeros in M. Instead, each synapse possesses a conductivity measured in siemens. While zeros in M can be mapped to a synapse with zero conductivity (corresponding to no synapse at all), the conductivity w representing a "one" in M needs to be carefully selected.

Even if we had to implement a BiNAM as a firing-rate neural network with fixed non-linearity, the threshold could be easily implemented with a sub-network trained for this task via back-propagation. Such a simple training of spiking neural networks is not possible.

Another problem concerns the threshold in the BiNAM recall rule from Equation (2.28): whereas the McCulloch-Pitts neuron model provided a perfect threshold function, single neurons in a spiking neural network do not offer such a well-behaved transfer function. The transfer function is specified as dynamical system which can only be affected by neuron parameter adaptation. This selection of neuron parameters is non-trivial.

In Section 3.1.4, we discuss the neuron behaviour required for a functioning BiNAM, given the data representation and the general network topology from the previous sections. We could then try to tune the neuron parameters and synapse weight w with respect to these rules. This endeavour is the topic of the next chapter 4.

3.1.1 *Input-/output spike sequences*

Classical feed-forward firing-rate coded neural networks do not possess time-dynamics: they merely describe a mathematical function f which maps vectorial input $\vec{x}_k$ onto output $\vec{y}_k$. The network does not possess any hidden state that may be influenced by previous computations, that is, there are no side-effects. Individual operations – for example a recall in an associative memory – are strictly separated from each other. When dealing with time-series of data, an evaluation of f can usually be executed within a single simulation time step.

Of course, separation between individual samples could be forced by repeatedly resetting the simulation. Yet, this may neither be practical (the network state may be important), nor possible (neuromorphic hardware).

Spiking neural networks do not share these properties: there exists no direct mapping between input and output; both are represented by a stream of continuous spike times, with each spike influencing the state of the involved neurons over time. For closed-loop operation this asynchronicity is one of the defining properties of a spiking neural network. For the sample based performance evaluation of our memory, we need to group input and output spikes that belong to the same sample $(\vec{x}_k, \vec{y}_k)$. As network simulations and especially neuromorphic

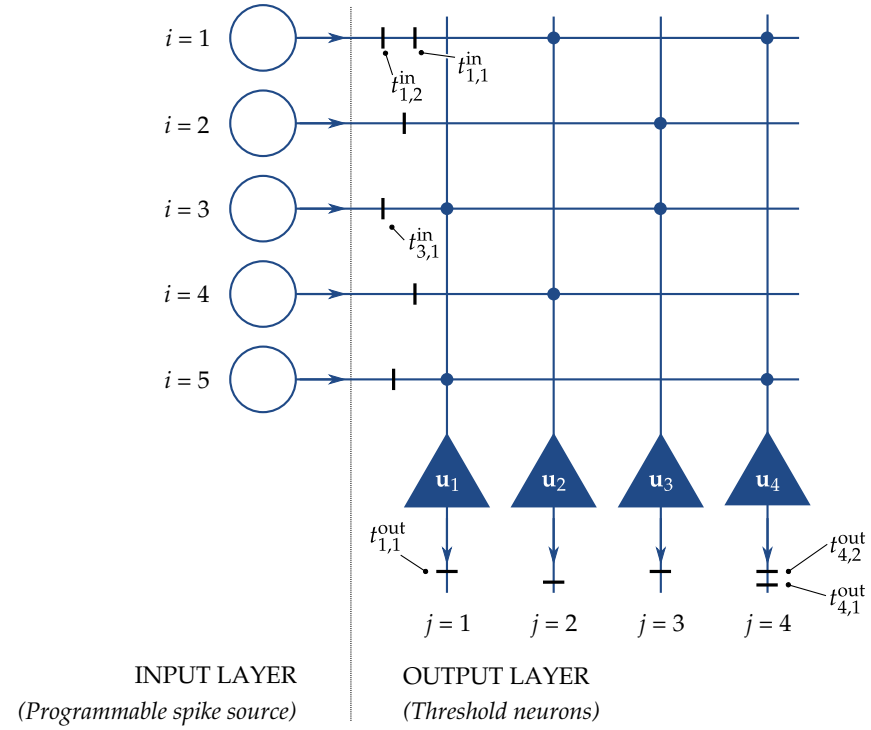


Figure 3.2: Test environment for the spiking implementation of a 5×4 BiNAM. The bold marks represent input/output spikes with examples of the spike train nomenclature in Equation (3.1).

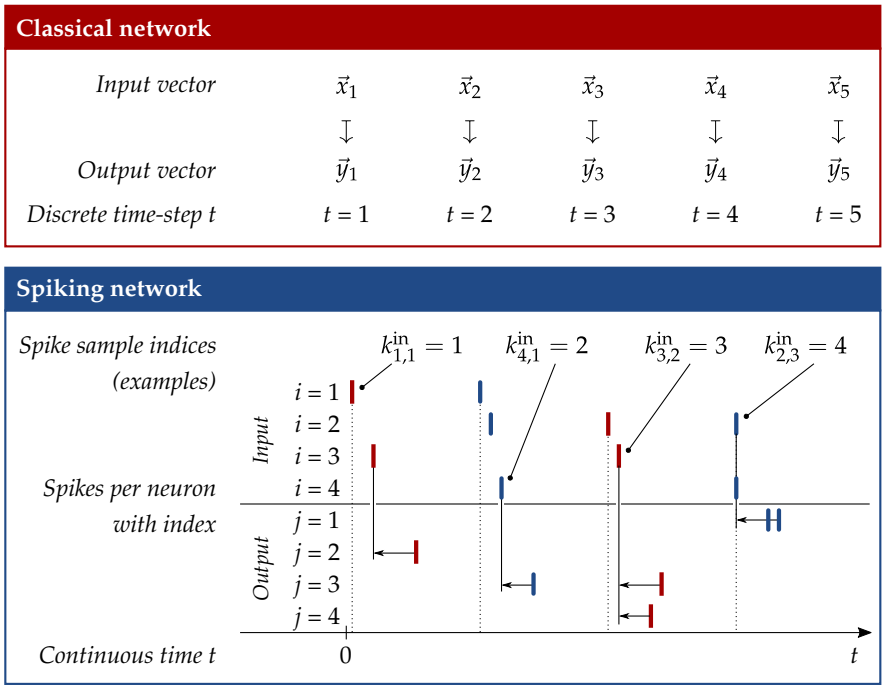


Figure 3.3: Conceptual comparison between the operation of a classical and spiking neural network: the classical neural network maps between discrete input and output vectors $\vec{x}_k \mapsto \vec{y}_k$. Such a mapping has to be artificially created on the input/output of a spiking network by assigning output spikes to the closest past input spike. Dotted vertical lines indicate the first spike belonging to a new sample.

hardware systems do not (or at least not consistently across platforms) allow to dynamically inject spikes during runtime, the entire input spike stream has to be assembled ahead of time and the network output has to be matched in a post-processing step.

To describe our solution to this problem, we first need to specify the spiking BiNAM test environment (Figure 3.2): an input layer, a group of m programmable spike sources, emulates external input to the network. Each source represents a single component i in $\vec{x}_k$ and produces a pre-programmed spike train t_i^{in}, which can be described as an ordered sequence of spike times $t_{i,\ell}^{\text{in}}$

$$t_i^{\text{in}} = (t_{i,1}^{\text{in}}, t_{i,2}^{\text{in}}, \ldots) \quad \text{with} \quad t_{i,\ell}^{\text{in}} \leq t_{i,\ell'}^{\text{in}} \Leftrightarrow \ell \leq \ell' \,. \tag{3.1}$$

During test execution output spikes t_j^{out} are recorded from the output layer, which consists of n neurons performing the actual thresholding operation. Analogously to the input spike times, $t_{j,\ell}^{\text{out}}$ denotes the ℓ-th spike for the j-th output component $\vec{y}_k$.

In order to assemble t_i^{in}, input samples $\vec{x}_k$ are converted into their spike time representation (discussed in Section 3.1.2). Let $k_{i,\ell}^{\text{in}}$ denote the sample index each spike $t_{i,\ell}^{\text{in}}$ belongs to. As depicted in Figure 3.3, a simple approach for assigning output spikes $t_{j,\ell}^{\text{out}}$ to the sample k they are most likely associated to, is by selecting $k_{j,\ell}^{\text{out}}$ as the sample index of the latest input spike issued before $t_{j,\ell}^{\text{out}}$

$$k_{j,\ell}^{\text{out}} = k_{i,\ell'}^{\text{in}} \quad \text{with} \quad i, \ell' = \arg\min_{i,\ell'} \{ t_{j,\ell}^{\text{out}} - t_{i,\ell'}^{\text{in}} \mid t_{j,\ell}^{\text{out}} - t_{i,\ell'}^{\text{in}} > 0 \} \,. \tag{3.2}$$

In case the network utilises the already mentioned "neuron population"-topology from Section 3.1.3, there may be multiple neurons for a single input/output component. For sake of simplicity regarding the above equations, we assume that spikes from multiple neurons for the same component j are fused into a single, virtual spike train.

Propagation of a signal through the network is usually not instantaneous. Thus, it can happen that an output spike is mapped to an input spike that already belongs to a later sample. If the minimum delay ζ between the first input and output spike of a sample is known, the output spike train can be virtually shifted back by ζ for the calculation of $k_{j,\ell}^{\text{out}}$ according to Equation (3.2)

$$i, \ell' = \arg\min_{i,\ell'} \{ t_{j,\ell}^{\text{out}} - \zeta - t_{i,\ell'}^{\text{in}} \mid t_{j,\ell}^{\text{out}} - \zeta - t_{i,\ell'}^{\text{in}} \geq 0 \} \,. \tag{3.3}$$

As shown in Figure 3.4, this potentially allows to pack input samples tighter without risking misclassification due to output spikes already being matched with the next sample. The technique can be used to implement pipelining of the memory recall operations. Note however, that due to the statefulness of the network, a higher input sample rate will affect the output.

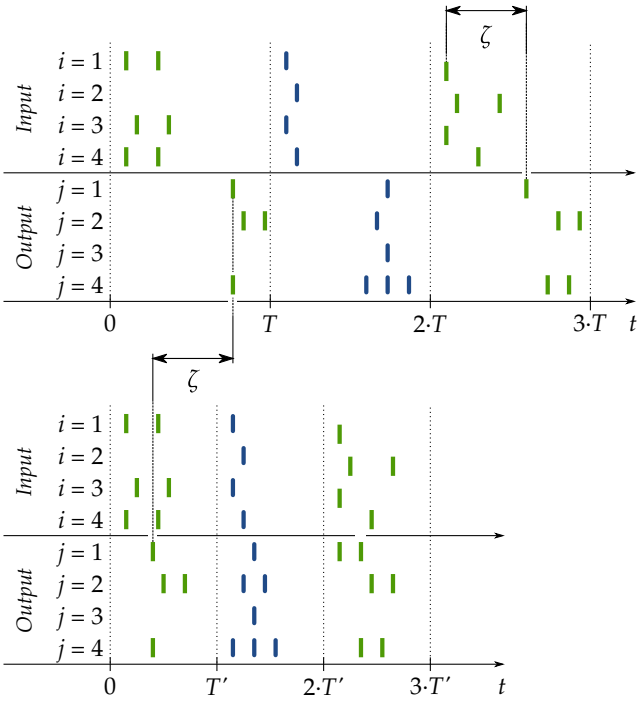


Figure 3.4: Example of input sample pipelining: virtually shifting the output spike times back by ζ allows higher input sample rate T' while preserving the spike-sample matching (colour coded). Note how the higher sample rate affects the output.

3.1.2 *Data encoding and input noise parametrisation*

We now need to decide on how to encode a single one-bit as a time sequence of spikes. In case the memory receives input from other networks – or artificial test data – which follows this specification, we expect our network to respond with another valid output spike pattern in this encoding. A straight-forward approach is to represent each one-bit in $(\vec{x}_k, \vec{y}_k)$ as a single spike. So in order to recall a vector for input $\vec{x}_k$ from the memory, we send a spike to all inputs i with $(\vec{x}_k)_i = 1$ at a certain time $t = T \cdot k$, where the constant T is the time between two input samples. The output component $(\vec{y}_k)_j$ is set to one, if an output spike with assigned sample index $k^{\text{out}}_{j,\ell} = k$ exists.

We expand this data generation scheme in two ways: by representing ones with potentially more than a single spike, resulting in *bursts*, and by modelling time noise, *jitter*, in the input.

Bursts Neurons in the cortex often operate in a *phasic bursting* mode in which action potentials occur in groups of multiple spikes (Figure 3.5), henceforth called *bursts* [CG90]. From a theoretical point of view signalling events using bursts has at least two advantages: they increase the robustness in case of noise – for example if a single spike

out of multiple is lost (as it might happen in both biological and neuromorphic hardware systems), the information about the corresponding event is not irrecoverably destroyed, and if false-positive single spikes are received by a neuron, these can be filtered out [Lis97]. Secondly, bursts allow to encode analogue values, either by variation of the spike count or the spike frequency within the burst.

To interface with "bursting" networks our associative memory implementation should support arbitrary integral burst sizes s^{in} and $s^{\text{out}} \geq 1$ chosen at design time. The network has to accept bursts of size s^{in} at the input, but it must also produce s^{out} spikes for each one-bit in the output.

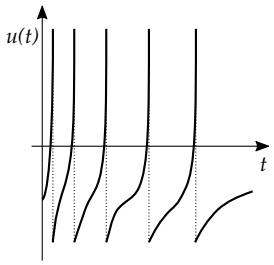


Figure 3.5: Sketch of a neuron membrane potential over time showing spikes from a burst with decaying interspike interval Δt.

When generating artificial bursts as test input, we parametrise these with an equidistant interspike interval (ISI) Δt. Note that this is only a coarse model of actual data that might be encountered in a large network, where the interspike interval may be inconsistent: for example in the AdEx model, the interspike interval is likely to increase over time (Section 2.3.6 and Figure 3.5). For small s^{in} this effect should be relatively subtle and selecting a constant Δt as the average interspike interval is a reasonable approximation.

On the output side, bursts imply fractional output values whenever a number of spikes that differs from the expected burst size s^{out} is generated. We redefine the value of the output component $(\vec{y}_k)_j$ as the quotient of the number of actual number of spikes encountered in the output time window, divided by the expected burst size s^{out}

$$(\vec{y}_k)_j = \frac{|\{k^{\text{out}}_{j,\ell} = k\}|}{s^{\text{out}}}\,. \tag{3.4}$$

To cope with arbitrary "fractional bit values" in the theoretical BiNAM storage capacity measure, we introduce an adapted version of the corresponding storage capacity equation (2.37) in Section 3.2.1.

False negative and positive input spikes In Section 2.5.6, we discussed missing and additional one-bits in the input vectors, where the corresponding events were modelled with probabilities p_0 and p_1. The representation of one-bits as a series of s^{in} spikes allows to insert and remove individual spikes instead of bits. Thus p_0 is used to describe the probability of skipping a single spike from a burst representing a "one", while p_1 describes the probability to issue a single spike from a virtual burst when encoding a "zero".

For large burst sizes s^{in} the probability of an entire input burst being removed or added is relatively small (p_0 and p_1 to the power of s^{in}). This is intentional, as the effects of entire bursts being removed and added can be well studied within the theoretical BiNAM framework (Section 2.5.6). For the spiking implementation we want to focus on noise concerning individual spikes instead.

Jitter Due to the involved dynamic systems, it is implausible for spiking neural networks to generate perfect timings. In order to emulate

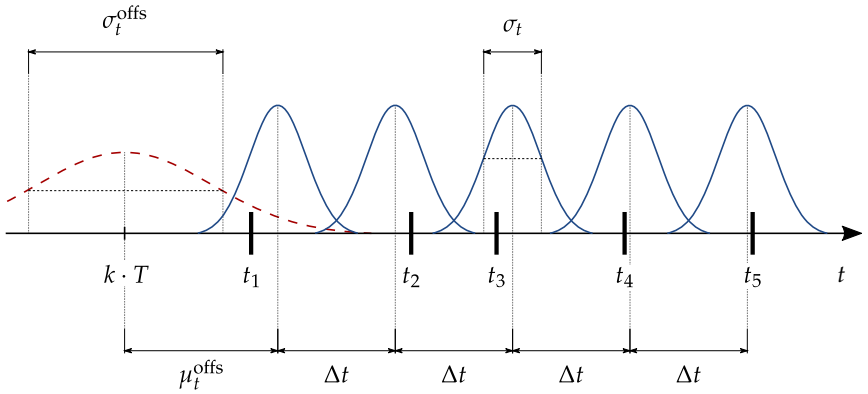


Figure 3.6: Parametrisation of an example input spike burst of size $s^{\text{in}} = 5$ with jitter as described in Equation (3.5). Bold marks correspond to the generated spike times $t_1, \ldots, t_5$. The bell-shaped functions visualise the Gaussian distributions from which the time offset μ_t^{offs} (dashed) and the individual spike times t_i (blue) are sampled.

these imperfections, we add small time offsets, *jitter*, to the spikes. In our model we sample a random offset from a Gaussian distribution with standard deviation σ_t for each spike time $t_{i,\ell}^{\text{in}}$. Additionally, we shift the entire burst by μ_t^{offs} in order to model synaptic or spatial delays. Again, μ_t^{offs} is sampled from a Gaussian distribution with standard deviation σ_t^{offs}.

Data show that cortical neurons can reproduce spike patters with millisecond precision under laboratory conditions [MS95].

An input burst for the k-th sample can thus be described as a sequence of s^{in} random variables $(t_1, \ldots, t_{s^{\text{in}}})$

$$t_\ell \sim \mathcal{N}\Big(k \cdot T + \mu_t^{\text{offs}} + \Delta t \cdot (\ell - 1), \sigma_t\Big) \text{ with } \mu_t^{\text{offs}} \sim \mathcal{N}(0, \sigma_t^{\text{offs}}) \,. \quad (3.5)$$

As time offsets are drawn from a Gaussian distribution, it might hold $t_{\ell-1} \geq t_\ell$. To allow for computationally efficient processing, a generated sequence of spike times should be sorted in a software implementation.

Note that the offset μ_t^{offs} is not an independent random variable for each t_i. Instead, it is sampled once for all spike times t_i within the burst.

A diagram showing the involved variables is given in Figure 3.6, along with an overview of all discussed parameters in Table 3.1 and an algorithm for the generation of an input spike sequence for a single input bit $(\vec{x}_k)_i$ in Algorithm 3.1.

3.1.3 *Neuron populations*

Findings regarding neural codes in the cortex suggest that neuron spike patterns can be modelled by a Poisson process [SK93]. This implies that single spike times convey little information – the instantaneous activity of neural circuitry can only be estimated when pooling responses from multiple neurons. It thus seems likely that neural networks are built redundantly, and information is encoded in the correlation of spike patterns originating from a *population* of neurons [SN94].

The Poisson-distribution

$$P_\lambda(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

describes the probability of an event to occur k-times in a certain time interval, given its average rate is λ.

With respect to our scenario, the following changes to the network topology take these considerations into account: we replace each neu-

NETWORK AND INPUT/OUTPUT PARAMETRISATION	
	Network and data parameters
m, n	Number of input and output components.
c, d	Number of "ones" in each input/output sample.
N	Number of input samples.
K	Number of neurons representing each output component.
	Output data parameters
s^{out}	Output burst size: number of spikes produced by a single neuron in an output burst which represents a "one". For neuron populations ($K > 1$), each individual neuron is expected to produce this number of spikes.
	Input data parameters
s^{in}	Input burst size: number of spikes sent in an input burst to a single neuron. For populations, each individual neuron receives K input bursts of this size.
T	Average time between input samples.
Δt	Equidistant interspike interval between input spikes in a burst.
	Input data noise parameters
σ_t	Standard deviation of the Gaussian distribution the input spikes are selected from.
σ_t^{offs}	Standard deviation of the random variable from which the spike burst offset is chosen. Models synaptic or spatial signal delays.
p_0	Probability of skipping a spike (false-negative) in an input burst when generating a "one".
p_1	Probability of adding a superfluous (false-positive) spike from a virtual input burst when generating a "zero".

Table 3.1: Summary of network, input/output and input noise parameters.

ron u_j with a population of K neurons $u_j^1, \ldots, u_j^K$, in which each fires a small number of s^{out} spikes per one-bit in the output component $(\vec{y})_j$. These neurons receive the same input and all output signals $y_j^1, \ldots, y_j^K$ are fed in parallel to the next input. The signals are treated as a single output y_j for evaluation purposes. As the total number of output spikes for an output component is now given as $K \cdot s^{\text{out}}$, a relatively small burst size can be used (possibly $s^{\text{out}} = 1$). Analogously to Equation (3.4), the value of the output component j can be calculated as

$$(\vec{y}_k)_j = \frac{|\{k_{j,\ell}^{\text{out}} = k\}|}{K \cdot s^{\text{out}}} \,. \tag{3.6}$$

Besides biological plausibility this architecture has two other potential advantages: tuning the neuron parameters in such a way that they produce a small number of spikes might turn out to be easier to accomplish than selecting neuron parameters which cause the neuron to

function GENERATEINPUT($M, N, K, s^{\text{in}}, T, \Delta t, \sigma_t, \sigma_t^{\text{offs}}, p_0, p_1$)
  init $t^{\text{in}}_{m,k} \leftarrow () \; \forall \; m \in \{1, \ldots, M\}, k \in \{1, \ldots, K\}$
  for $n \leftarrow 1$ **to** N **do** ▷ Iterate over all samples
    for $m \leftarrow 1$ **to** m **do** ▷ Iterate over all input components
      if $(\vec{x}_n)_m = 1$ **then** ▷ Set spike omission probability p
        $p \leftarrow p_0$
      else
        $p \leftarrow 1 - p_1$
      end if
      for $k \leftarrow 1$ **to** K **do** ▷ Iterate over all neurons in the population
        $t^{\text{offs}} \leftarrow \text{NORMAL}(n \cdot T, \sigma_t^{\text{offs}})$ ▷ Select burst time offset
        for $\ell \leftarrow 1$ **to** s^{in} **do** ▷ Iterate over all spikes
          if RANDOMSELECT($[0, 1)) \geq p$ **then**
            $t^{\text{in}}_{m,k} \leftarrow t^{\text{in}}_{m,k} \| \; \text{NORMAL}(t^{\text{offs}} + (\ell - 1) \cdot \Delta t, \sigma_t)$
            $k^{\text{in}}_{m,k} \leftarrow k^{\text{in}}_{m,k} \| n$
          end if
        end for
      end for
    end for
  end for
  $k^{\text{in}}_{m,k} \leftarrow \text{SORTBYKEY}(k^{\text{in}}_{m,k}, t^{\text{in}}_{m,k}) \; \forall m, k$ ▷ Sort indices by spike times
  $t^{\text{in}}_{m,k} \leftarrow \text{SORT}(t^{\text{in}}_{m,k}) \; \forall m, k$ ▷ Sort spike times
  return $t^{\text{in}}, k^{\text{in}}$
end function

Algorithm 3.1: Input spike train generation algorithm. Generates the input spike trains t^{in} and indices k^{in} for all neurons according to the data parameters in Table 3.1. For a more consistent notation inside the algorithm, the input dimensionality is denoted as M instead of m.

reliably produce an exact large number of output spikes. Furthermore, the redundancy introduced by the neuron populations allows single neurons to fail without severely impacting the network performance.

On the downside, the number of required neurons scales linearly with K, the number of required synapses scales quadratically with K, inevitably increasing both size and energy consumption of the network. An example of the topology for $K = 2$ is sketched in Figure 3.1(b) on page 39.

3.1.4 *Required neuron behaviour*

We have discussed two possible BiNAM spiking neural network topologies. We now need to specify the neuron behaviour required for the operation of the network. Basically, there are two conditions each neuron in the network must fulfil: the *threshold condition* and the *reset condition*.

Threshold condition As specified in the BiNAM recall rule in Equation (2.28), the neuron needs to fulfil a thresholding behaviour: let n_{in} denote the number of input spikes within an input time frame of approximately $\Delta t \cdot s^{\text{in}}$. The number of output spikes the neuron should produce in response to this input is then given as

$$n_{\text{out}}(n_{\text{in}}) = \begin{cases} s^{\text{out}} & \text{if } n_{\text{in}} \geq n_1 = c \cdot s^{\text{in}} \cdot K \\ 0 & \text{if } n_{\text{in}} \leq n_0 = (c-1) \cdot s^{\text{in}} \cdot K \end{cases} . \tag{3.7}$$

For input spike counts in the interval (n_0, n_1), the neuron behaviour is deliberately left undefined, as these values should not occur under perfect conditions and a sharp threshold such as $n_0 = n_1 - 1$ would be very demanding to realise.

It should also be stressed, that the neuron parameters for the realisation of the threshold behaviour can only be chosen for a constant time frame $\Delta t \cdot s^{\text{in}}$. At some point, if input spikes arrive in a shorter or longer period of time, more or fewer output spikes will be certainly generated due to the fixed neuron time dynamics. However, we can try to maximise the tolerance to deviations from $\Delta t \cdot s^{\text{in}}$ until a violation of the threshold condition in Equation (3.7) occurs.

Reset condition This condition is closely linked to the way in which we test the network and match output spikes to the input events (Section 3.1.1): a new sample k with input vector $\vec{x}_k$ is presented to the network in a fixed interval determined by T. All output spikes are interpreted as a response to the sample k to which the last input spike belonged. Correspondingly, the neuron dynamics must ensure that the neuron (a) does not fire any additional output spikes as a response to an input that is longer than T ago and (b) is guaranteed to approximately reach its initial condition after a period T has passed – otherwise input samples would influence later tests in an unwanted and uncontrolled way.

3.2 MEMORY EVALUATION MEASURES

Note that the presented measures are potentially contradict each other: for example, a network with low energy consumption may exhibit a high latency and small robustness.

The previous section described the network topology, overall test methodology and two abstract conditions the individual neurons need to fulfil. In this section we discuss various measures for the evaluation of the overall performance of the spiking associative memory implementation: storage capacity, robustness in case of noise, latency and energy consumption.

3.2.1 *Storage capacity*

For a classical BiNAM the storage capacity evaluation measure has been defined in Equation (2.37) on page 33 as entropy of the output

samples, minus the entropy of false positives and negatives for each sample k. As the binomial coefficient used in the formula classically describes a combinatorial process, it is only defined for integral false positive and negative counts n^k_{fp} and n^k_{fn}. However, in the spiking neural network implementation, those values are derived from fractional bit values $(\hat{y}_k)_j$ and are neither integral nor limited to the range expected by the storage capacity formula. The pragmatic solution would be to round the bit values to zero and one. Yet this approach would discard useful information about slight performance degradations.

We thus need to define how to adapt the entropy equation for non-integral values and how to map fractional bit values onto range-limited false positive and negative counts.

The first requirement can be fulfilled by replacing the faculty operations hidden inside the binomial coefficients with their complex-valued generalisation, the gamma function $\Gamma(x+1) = x!$. The real-valued generalisation of the binomial coefficient is defined as

An approximation of $\ln(\Gamma(x))$ is provided by most programming environments as a single, numerically stable function.

$$\begin{aligned}\operatorname{lb}\binom{n}{k} &= \operatorname{lb}\left(\frac{n!}{n!\cdot(n-k)!}\right) = \operatorname{lb}\left(\frac{\Gamma(n+1)}{\Gamma(k+1)\cdot\Gamma(n-k+1)}\right)\\ &= \operatorname{lb}(\Gamma(n+1)) - \operatorname{lb}(\Gamma(k+1)) - \operatorname{lb}(\Gamma(n-k+1))\,. \end{aligned} \tag{3.8}$$

Given the generalised binomial coefficient we can use the original entropy equation (2.37) with fractional error counts. These are calculated by clamping the fractional bit values to one and linearly accumulating the distance between actual output $(\hat{y}_k)_j$ and expected output $(\vec{y}_k)_j$:

Multiplying n and k in the binomial coefficient by s^{out} seems like another solution to eliminate fractional values. This would imply that the fractional values carry additional information. Strictly speaking this is true, but we never use this information. Our sole goal is to preserve early signs of performance improvement or degradation in the storage capacity measure.

$$n^k_{\text{fp}} = \sum_{j:(\vec{y}_k)_j=0} \min\{1, (\hat{y}_k)_j\} \qquad n^k_{\text{fn}} = \sum_{j:(\vec{y}_k)_j=1} 1 - \min\{1, (\hat{y}_k)_j\} \tag{3.9}$$

With these adaptations the storage capacity measure can be used in conjunction with spiking BiNAM implementations. In case the neurons fulfil the conditions listed in Section 3.1.4, the storage capacity measure will return the same values as in the theoretical case, which potentially limits the discriminatory power of the measure. As differentiating between various environments is the primary objective of a benchmark, the discriminatory power might be increased by injecting artificial noise into the simulation and thus obtaining information about the robustness of the current parameter set.

3.2.2 *Robustness in case of noise*

When comparing the robustness of two associative memory implementations in case of noise, we can measure the impact of a certain noise parameter η on another evaluation measure – for example the storage capacity – under variation of η. In the remaining section we specify which kind of noise can be addressed and how it is parametrised for testing.

Input noise Associative memories are usually designed around *additive/subtractive input noise*: given an arbitrary input vector $\vec{x}$, the associative memory should still be capable of returning the output $\vec{y}_k$ for the sample k with $\|\vec{x}_k - \vec{x}\|$ minimal. The robustness in case of this kind of noise can be measured by varying the input noise parameters p_0 and p_1. Another kind of input noise, *jitter* can be controlled as noise parameters σ_t and σ_t^{offs}, which control random offsets in the spike timings and randomly model delays in pre-synaptic networks, as well as synaptic or spatial delay. Both noise parameters were described in Section 3.1.2.

The iterative digital-to-analogue conversion process used in the NM-PM1 hardware system is an additional source for non-deterministic noise.

Parameter noise In biological and analogue neuromorphic hardware systems, there are slight deviations in the circuitry which result in two neurons or synaptic connections with supposedly equal configuration to behave differently. In systems based on numeric integration artefacts are introduced by quantisation.

Both deviations can be modelled by artificially adding Gaussian noise to the neuron parameters and synaptic weights: upon network generation we sample a small noise term η from a Gaussian distribution with standard deviation σ_ϕ and add it to the given value for the parameter ϕ. This operation is performed for each neuron and every synaptic connection individually. Care must be taken to clamp the noisy parameter values to their valid range.

State noise This term refers to noise present in the state variables (membrane potential, synapse conductivities, adaptation current in AdEx) of each neuron. In biological systems, spatially close synapses can affect each other although they are not directly connected [BH97]. In analogue neuromorphic hardware systems, both thermal noise and crosstalk between neighbouring circuits may superimpose noise onto the neuron state. In software implementations quantisation artefacts can be interpreted as noise superimposed on the state signal (cf. Signal-to-quantisation-noise ratio (SQNR) [LD09]). Quantisation artefacts are especially relevant in the many-core neuromorphic hardware system which – in its current version – is based on fixed-point numbers with limited resolution.

Artificially injecting this kind of noise into a neural network simulation is not well supported by the software toolchain, so we are unable to analyse the impact of state noise. It should however be kept in mind that this kind of noise exists. The previously mentioned noise models are thus incomplete and cannot be used as the sole explanation of neuromorphic hardware system results.

3.2.3 *Latency and throughput*

For conventional digital systems, *latency* describes the time required for the result of an issued operation to be available as output, whereas

throughput describes the rate new operations can be issued at.

A rather simplistic approach to measuring latency is the following: let $\hat{t}_k^{\text{in}}$ denote the time of the last input spike and $\hat{t}_k^{\text{out}}$ the time of the last output spike associated to a sample k (given that such an output spike exists)

$$\hat{t}_k^{\text{in}} = \max_{i,\ell}\{t_{i,\ell}^{\text{in}} | k_{i,\ell}^{\text{in}} = k\}\,, \qquad \hat{t}_k^{\text{out}} = \max_{j,\ell}\{t_{j,\ell}^{\text{out}} | k_{j,\ell}^{\text{out}} = k\}\,. \tag{3.10}$$

The latency for a single sample δ_k is now defined as

$$\delta_k = \hat{t}_k^{\text{out}} - \hat{t}_k^{\text{in}}\,. \tag{3.11}$$

The above definition has two problems. δ is only well-defined if an output is produced for an input k. Furthermore, the actual throughput of the system might be larger than implied by δ, since the neuron parameters were chosen such that the neuron is ready for the next input sample within a time window T. As by definition $T > \delta$, the effect of a low latency δ could be nullified by the delay imposed by T until a next sample can be processed.

A combined measure of latency and throughput is *critical time window* analysis. Instead of presenting samples in the nominal interval T, successively smaller T' are tested, until the measured storage capacity drops below a relative threshold p of the original storage capacity. The corresponding time window length is called T_p^{crit}. It corresponds to the maximum frequency at which the memory can be operated without significant impact on its storage capacity (Figure 3.7). This approach can of course be combined with the pipelining from Section 3.1.1.

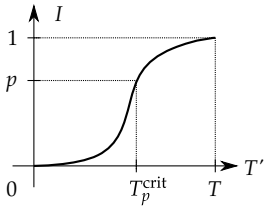


Figure 3.7: Sketch of the critical time window measure: T_p^{crit} is defined as sample interval T' at which the relative storage capacity I drops below p.

Certain memory usage patterns might cause interference, which causes the value of T_p^{crit} to depend on the input sample order – however for large sample counts N this problem should be negligible.

3.2.4 *Energy*

Another common evaluation measure for any electronic system is its energy consumption: lower energy consumption implies less heat being dissipated, possibly allowing for higher packing density and overall lower cost of operation. The following considerations should be taken into account for measuring the energy consumption of the associative memory.

For a numerical simulation on a traditional computer architecture, power consumption is mostly influenced by the required simulation time. In analogue neuromorphic hardware however, power consumption is directly related to the neuron state. While modelling the energy consumed by an analogue neuron is out of the scope of this thesis, the required energy J can be roughly modelled as

$$J = \int_{t=0}^{\infty} (E_{\text{L}} - u(t)) \cdot i(t)\,\mathrm{d}t\,, \tag{3.12}$$

where $i(t)$ are the accumulated currents flowing through the neuron membrane.

3.3 DATA GENERATION

In the previous sections we have established the foundations for the realisation of a BiNAM as spiking neural network: we have provided the network topology, a representation of the input data, high-level conditions the spiking neurons need to fulfil and a set of evaluation measures which allow to evaluate the performance of the associative memory. The last missing piece is to describe the actual (non-spiking) data that should be used to test the BiNAM.

It is important to stress that we do not try to benchmark the BiNAM itself – the theoretical limits of this particular associative memory have already been thoroughly studied over the last forty years. Instead, we test a set of particular BiNAM implementations. Generated data should thus utilise the BiNAM to the full instead of modelling datasets that may be found in biological systems.

Section 3.3.1 summarises the test data meta parameters. The most obvious candidate for test data – uncorrelated, random bit vectors – is discussed in Sections 3.3.2 and 3.3.3. We then discuss *balanced* datasets, which evenly occupy the BiNAM and an algorithm which generates such data in Sections 3.3.4 and 3.3.5.

3.3.1 *Dataset parametrisation*

A dataset $\mathfrak{D} = \{(\vec{x}_k, \vec{y}_k)\}$ without duplicate input vectors $\vec{x}_k$ as defined in Equation (2.26) is parametrised by the number of samples N, the dimensionality of the input and output vectors (m, n respectively) and the number of bits set to one in those vectors: $\|\vec{x}_k\|_1 = c$ and $\|\vec{y}_k\|_1 = d$. The storage capacity formula for hetero-association given in Equation (2.37) requires the input and output data to be neither inter- nor intracorrelated. Input and output vectors can be represented as a $N \times m$ matrix X and a $N \times n$ matrix Y.

While the perfect test data would produce a minimum number of possible errors while still fulfilling the uncorrelatedness condition, generating such data would require to solve a large binary linear programming problem, which in the general case is NP-complete [Kar72]. A feasible approach is thus to simply generate random input and output vectors. We discuss the expected behaviour of random data and its generation in the next two sections, followed by a more sophisticated data generation algorithm which ensures uniform usage of the network even for small memory sizes at the cost of violating the uncorrelatedness rule.

3.3.2 *Expected behaviour in reaction to uncorrelated random data*

In order to analyse the expected behaviour of random data in the BiNAM we first need to understand the conditions under which errors

occur if non-noisy input data is given. As described in Section 2.5.6 the only kind of errors in the output of a theoretical BiNAM with perfect input are false positives. According to Equation (2.28), for an input $\vec{x}_k$ a false positive will arise at position j of the output (for which $(\vec{y}_k)_j = 0$) exactly if

$$(M^\top \cdot \vec{x}_k)_j = \sum_{i:(\vec{x}_k)_i=1} (M)_{ij} \stackrel{!}{=} \|\vec{x}_k\|_1 = c\,. \tag{3.13}$$

In other words: a false positive in output component j for a sample $(\vec{x}_k, \vec{y}_k)$ is generated if for each "one" at position i in $\vec{x}_k$ there exists a sample k' with $(\vec{y}_{k'})_j = 1$ and $(\vec{x}_{k'})_i = 1$ [Pal80]. The condition of an additional one occurring at position j in the output for $\vec{x}_k$ is henceforth abbreviated as $C(k, j)$. If we neglect that there may be no duplicate input vectors $\vec{x}_k$, the expected total number of false positives per sample $\langle n_{\text{fp}} \rangle$ can be easily calculated.

Let $\wp$ denote the probability of a single cell i, j in M being set to one after N samples have been trained

The probability $\wp$ is the same for all cells i, j since the bits are distributed independently and uniformly across the input and output vectors.

$$\begin{aligned}
\wp &= \Pr((M)_{i,j} = 1) = 1 - \Pr((M)_{i,j} = 0) && (3.14)\\
&= 1 - \prod_{k=1}^{N} \Pr((\vec{x}_k)_i = 0 \vee (\vec{y}_k)_j = 0) && (3.15)\\
&= 1 - \prod_{k=1}^{N} \left(1 - \Pr((\vec{x}_k)_i = 1 \wedge (\vec{y}_k)_j = 1)\right) && (3.16)\\
&= 1 - \prod_{k=1}^{N} \left(1 - \Pr((\vec{x}_k)_i = 1) \cdot \Pr((\vec{y}_k)_j = 1)\right) && (3.17)\\
&= 1 - \left(1 - \frac{c \cdot d}{m \cdot n}\right)^N . && (3.18)
\end{aligned}$$

If we wanted to respect the fact that there are no duplicates in the entirety of $\vec{x}_k$, we would not be allowed to express $\Pr((M)_{i,j} = 0)$ as a product: the individual events are no longer independent. However, the error introduced by this assumption is sufficiently small for large memories [Pal80].

According to Equation (3.13) the probability $\Pr(C(k, j))$ of a false positive occurring in the j-th component of sample k can be described as

$$\Pr(C(k,j)) = \Pr\Big((M^\top \cdot (\vec{x}_k))_j = c\Big) = \Pr\left(\sum_{i=1}^{m} (M)_{i,j} \cdot (\vec{x}_k)_i = c\right) \tag{3.19}$$

$$= \Pr\left(\sum_{i=1}^{m} \left(\bigvee_{\ell=1}^{N} (\vec{x}_\ell)_i \cdot (\vec{y}_\ell)_j\right) \cdot (\vec{x}_k)_i = c\right). \tag{3.20}$$

Since $(\vec{y}_k)_j = 0$ (otherwise a one in the j-th output component would be a *true* positive), we can add the condition $\ell \neq k$ to the inner "$\vee$"-operation of Equation (3.20) without changing its value

$$\Pr(C(k,j)) = \Pr\left(\sum_{i=1}^{m} \left(\bigvee_{\ell=1,\ell\neq k}^{N} (\vec{x}_\ell)_i \cdot (\vec{y}_\ell)_j\right) \cdot (\vec{x}_k)_i = c\right). \tag{3.21}$$

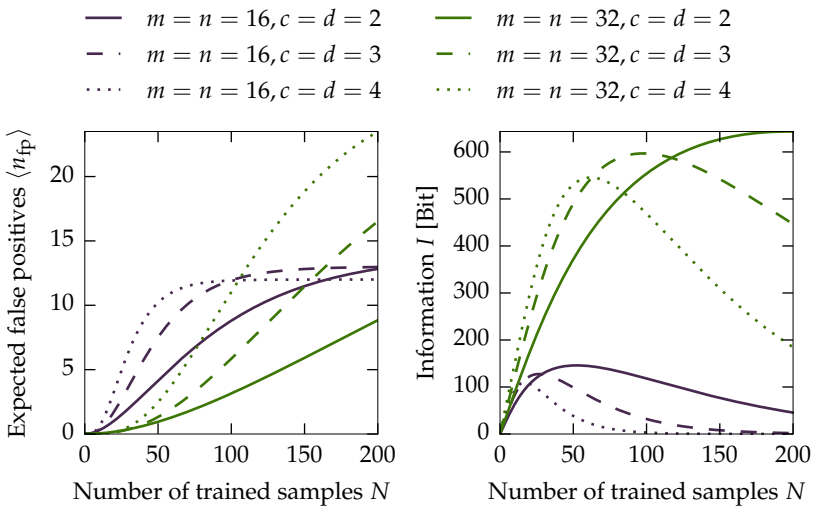


Figure 3.8: Estimated number of false positives per sample k and corresponding information I for varying memory size m, n and number of bits set c, d.

Now it is clearly visible that there is no dependence between M and $\vec{x}_k$ in Equation (3.19). The final $\Pr(C(k,j))$ is thus given as

$$\Pr(C(k,j)) = \Pr\left(\bigwedge_{i:(\vec{x}_k)_i=1} (M)_{i,j} = 1\right) = \wp^c = \left(1 - \left(1 - \frac{c \cdot d}{m \cdot n}\right)^N\right)^c . \tag{3.22}$$

For large m, n the error caused by allowing duplicates in X diminishes. A proof is given in appendix B of [Pal80].

The expected number of false positives $\langle n_{\text{fp}} \rangle$ per sample k for uncorrelated, randomly generated input and output vectors is consequently

$$\langle n_{\text{fp}} \rangle = (n - d) \cdot \left(1 - \left(1 - \frac{c \cdot d}{m \cdot n}\right)^N\right)^c . \tag{3.23}$$

Figure 3.8 shows $\langle n_{\text{fp}} \rangle$ and the corresponding information estimate for a varying number of samples N and different combinations of m, n, c, d.

3.3.3 *Random data generation algorithm*

Input and output vector matrices X, Y are independently generated as data blocks $B = (\vec{b}_1, \ldots, \vec{b}_N)^\top$, where each $\vec{b}_i$ is an r-tuple containing r_1 ones and $r_0 = r - r_1$ zeros

$$\vec{b}_i \in \mathfrak{B}^r_{r_1} = \left\{ (b_1, \ldots, b_r) \,\middle|\, b_j \in \mathbb{B}, \textstyle\sum_{j=1}^r b_j = r_1 \right\} . \tag{3.24}$$

According to basic combinatorics the size of $\mathfrak{B}^r_{r_1}$ is

$$|\mathfrak{B}^r_{r_1}| = \binom{r}{r_1} = \binom{r}{r - r_1} = \binom{r}{r_0} . \tag{3.25}$$

```
1: init B ← 0 ∈ 𝔹^{N×r}                                  ▷ Result matrix
2: for i ← 1 to N do
3:     for j ← r − r_1 + 1 to r do
4:         ℓ ← RandomSelect({1, ..., j − 1})  ▷ Uniformly select set entry
5:         if B[i, ℓ] = 1 then
6:             B[i, j] ← 1
7:         else
8:             B[i, ℓ] ← 1
9:         end if
10:    end for
11: end for
```

Algorithm 3.2: Algorithm for the generation of a block B of N uncorrelated random vectors of length r, containing exactly r_1 ones.

Each vector $\vec{b} \in \mathfrak{B}^r_{r_1}$ can be uniquely represented as set $\mathfrak{b}$ containing the indices of "ones" in $\vec{b}$ (Figure 3.9)

$$\mathfrak{b} = \{i \mid b_i = 1\} \qquad \mathfrak{b} \in \mathfrak{b}^r_{r_1} = \{\{i_1, \ldots, i_{r_1}\} \mid i_j \in \{1, \ldots, r\}\}\,. \tag{3.26}$$

$(1,0,1,0,0) \to \{0,2\}$
$(0,0,1,0,1) \to \{2,4\}$
$(0,1,0,0,1) \to \{1,5\}$

Figure 3.9: Elements of $\mathfrak{B}^5_2$ and their corresponding set representation.

The set $\mathfrak{b}^r_{r_1}$ is the set of all r_1-sized sets over $\{1, \ldots, r\}$, or – in mathematical terms – the set of all r_1-combinations of $\{1, \ldots, r\}$. As shown in Algorithm 3.2 a set of n random r_1-combinations can be efficiently generated in $\mathcal{O}(n \cdot r_1)$ time [BF87].

This algorithm does not ensure the absence of duplicates in the input. While the probability of a duplicate is sufficiently small for large m, one could expand the algorithm by a hash-table lookup and retry if a vector has already been generated. Yet such a method may not terminate if an exhaustively large dataset is generated. An alternative which is guaranteed to terminate is the algorithm presented in Section 3.3.5.

3.3.4 *Balanced data*

For practical experiments with relatively small networks, uncorrelatedness does not guarantee a uniform occupancy of the storage matrix bits, as the probability for two randomly generated samples to activate the same bits in M increases with smaller memories. This might be a problem when benchmarking neuromorphic hardware systems, where we would like to ensure that at any time during the test run all neurons and synapses get approximately the same workload in order to produce a more stable behaviour.

The introduction of the *balancing* condition – besides uncorrelatedness and uniqueness of the input vectors – allows to ensure this. For any $k' \leq N$ the column sums of a generated data block B (which may either refer to the input data X or the output Y) must be balanced.

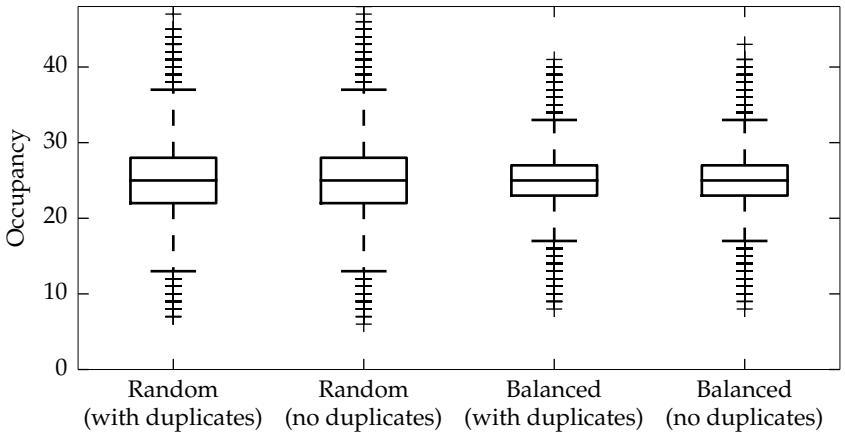


Figure 3.10: Comparison of the BiNAM matrix occupancy distribution for different data generation methods: the occupancy measures how many times a BiNAM matrix cell would be set to one during training. Training was simulated for hetero-association with 100 samples in a 32×32 BiNAM with $c = d = 16$ one-bits for each vector. The box plots visualise the distribution of the occupancy values for 1000 independent training runs over all matrix cells (the box extends from the lower to the upper quartile, with a line at the median, the whiskers depict the 1.5 interquartile range, crosses outliers).

Specifically, it must hold

$$\max_j \left\{ \sum_{k=1}^{k'} B_{kj} \right\} - \min_j \left\{ \sum_{k=1}^{k'} B_{kj} \right\} \leq 1 \, \forall k' \leq N \,. \tag{3.27}$$

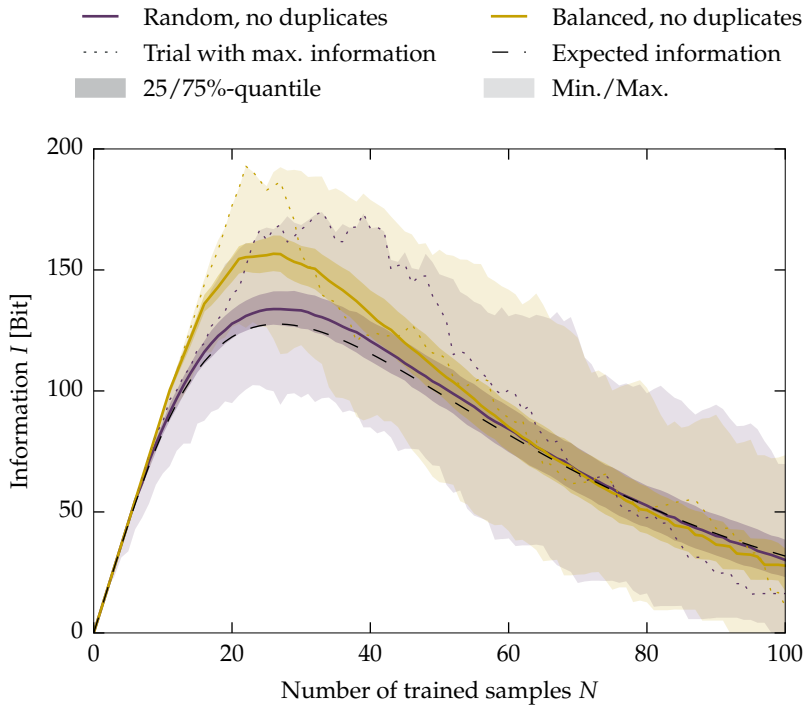
A potential effect of the balancing is shown in Figure 3.10: when calculating how many times a cell in the BiNAM is set to one during training – which can be computed as $X^T \cdot Y$ – the variance in occupancy significantly reduces when balancing is enabled.

An alternative interpretation of the balancing rule is a prolonged creation of false positives, which in return increases the information that can be stored in the BiNAM: input and output data balancing selects rows and columns in the BiNAM which have not yet been used as often as others, lowering the probability of the false-positive condition $C(r, j)$. An experiment showing this effect is depicted in Figure 3.12 – balanced vectors without duplicates for both input and output data achieves the highest BiNAM storage capacity.

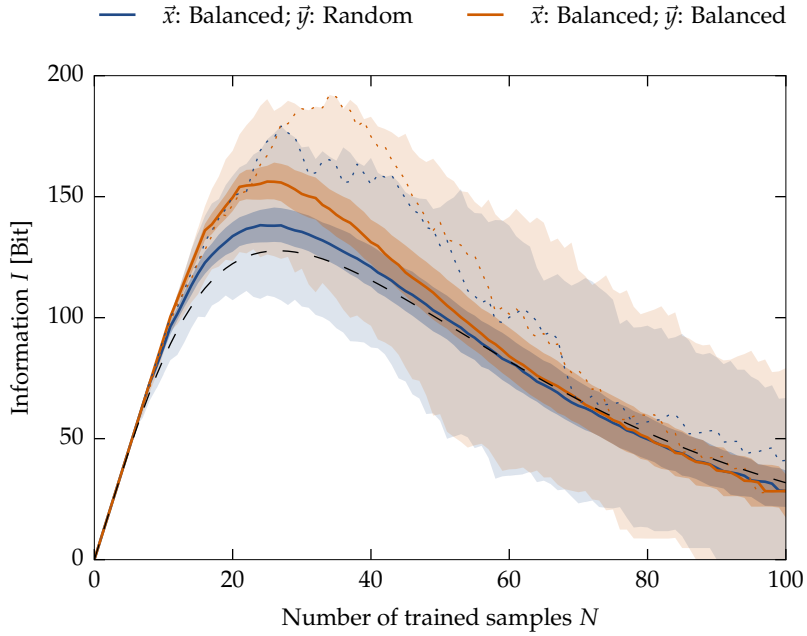
$\vec{b}_1 = (0,1,0,0,1,1)$
$\vec{b}_2 = (1,0,1,1,0,0)$
$\vec{b}_3 = (1,1,0,0,0,1)$
$\vec{b}_4 = (0,0,1,1,1,0)$

Figure 3.11: Correlation introduced by data balancing for $r = 6$ and $r_1 = 3$: while the data generation algorithm can randomly select $\vec{b}_1$ and $\vec{b}_3$, samples $\vec{b}_2$ and $\vec{b}_4$ are predetermined.

Unfortunately, these results must be taken with a grain of salt: from a strict mathematical point of view, balancing causes intracorrelation in a dataset B (Figure 3.11). Therefore the storage capacity formula from Equation (2.37) cannot be applied. However, we believe that the introduced correlation (similarly to requiring no duplicates in the input) is sufficiently small for any reasonably large number of samples N. After all, an algorithm generating balanced data could still generate all possible data vectors, though it limits the random selection in each step to those vectors which fulfil the balancing condition in Equation (3.27).

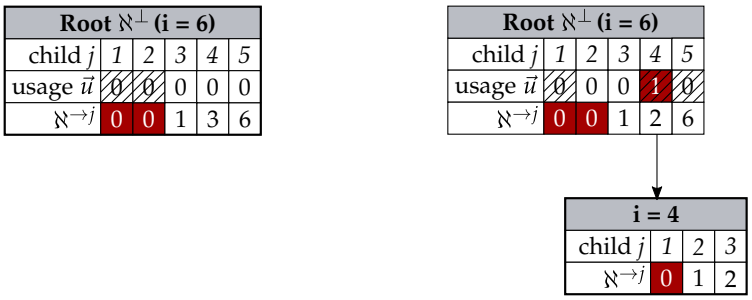


(a) Homogeneous data generation method. Both $\vec{x}$ and $\vec{y}$ are without duplicates.



(b) Heterogeneous data – $\vec{x}$ is without duplicates, $\vec{y}$ potentially with duplicates.

Figure 3.12: Random versus balanced data generation. 1000 trials in which a 16×16 BiNAM with 3 ones in input/output is incrementally trained with N samples were conducted. Samples are either generated randomly, or randomly with balanced bit allocation. The bold line depicts the median information, the dotted line the trial with maximum information and the dashed line the prediction according to Equation (3.23). The 25/75%-quantile and the minimum/maximum over all trials are shaded. (a) shows the experiment for input and output data homogeneously generated with one of the two methods, (b) explores the heterogeneous case.



(a) Root node in the initial state of the algorithm.

(b) Updated tree after the selection of the first "one" at index three.

Figure 3.13: Example of the data generation algorithm with $r = 5$ and $r_1 = 3$. The bold frame indicates the currently active node $\aleph$. In (a) none of the bits have been used yet. Indices 1 or 2 (hatched) can not be chosen as this would not leave a sufficient number of indices to choose from for the remaining bits. In (b) the index 4 has been selected: the usage count is incremented, the number of remaining permutations starting with 4 is decremented in the root node. Only bit indices 2 and 3 are viable next child indices for the new node.

3.3.5 *Balanced data generation algorithm*

A "prefix-tree" or "Trie" is a standard data-structure in computer-science used for compact storage of a set of sequences. Common prefixes are represented by a single node. Whenever the sequences diverge the Trie branches [Knu98].

Subsequently we present a simple and fast algorithm for the generation of N random and balanced bit vectors without duplicates. As in Algorithm 3.2, the method is based on the set representation $\mathfrak{b}_k$ of a bit vector $\vec{b}_k$, and the one-bit indices are chosen one at a time.

The key idea of the algorithm is to induce an order on the possible r_1-combinations by representing them as a sequence of one-indices $(i_1, \ldots, i_{r_1})$ with $i_j > i_{j'} \Leftrightarrow j' > j$. These sequences are organised in a prefix-tree – *Trie* – of depth $r_1 + 1$: each node represents the index of a single one-bit, every possible path from the root node to a leaf represents a unique combination. This properties allows to track how many not-yet-generated combinations are reachable from a node.

The root node $\aleph^{\perp}$ can be seen as a virtual node with index $i = r + 1$ at level $\ell = 0$.

For each Trie node $\aleph$ with index i at level ℓ we initially calculate a table containing the number of possible combinations for each child $j < i$ and $\tilde{r} = r_1 - \ell$ remaining ones

$$\aleph^{\to j} = |\mathfrak{b}^j_{\tilde{r}}| = \begin{cases} \binom{j}{\tilde{r}} & \text{if } j \geq \tilde{r} \\ 0 & \text{if } j < \tilde{r} \end{cases} . \tag{3.28}$$

Whenever a path along a child node $\aleph[j]$ is traversed, the corresponding table entry $\aleph^{\to j}$ is decremented, as a new combination with this path as prefix is being generated. Once $\aleph^{\to j}$ is zero, the corresponding path is ignored in future index selections (Figures 3.13 and 3.15).

For each of the N vectors, child indices j are randomly selected from the possible child indices, moving the current-node cursor $\aleph$ through the Trie. As soon as the lowest Trie level is reached after r_1 choices, the cursor is reset to the root $\aleph^{\perp}$ and the next vector is generated. An

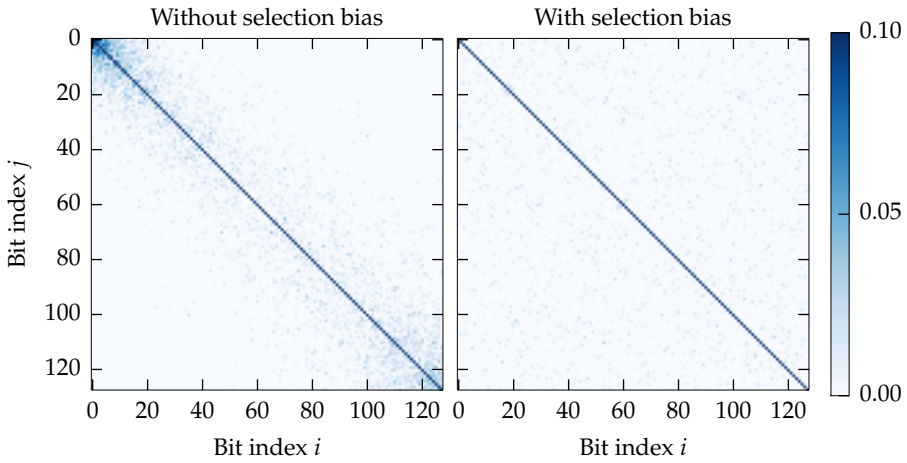
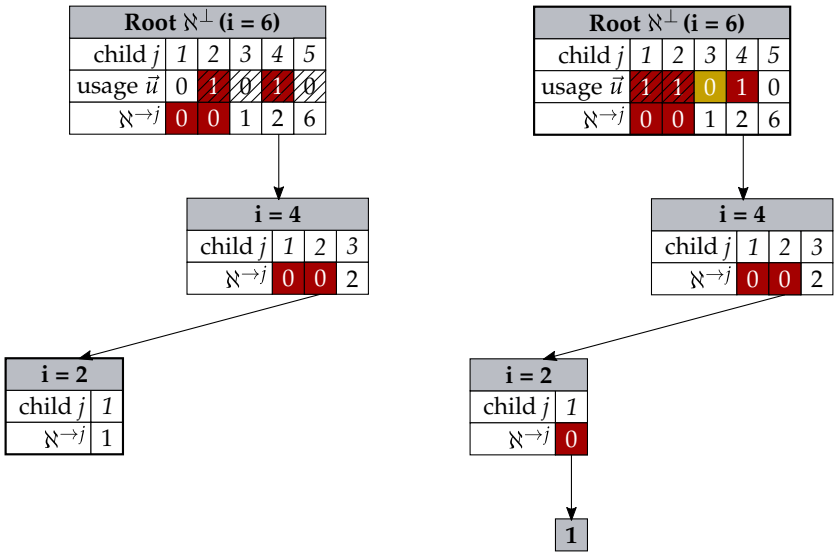


Figure 3.14: Linear correlation coefficient for two bits i, j in 10000 generated bit vectors with a length of $r = 128$ and $r_1 = 3$ ones. The left plot shows the result for data generated with disabled selection bias, the right with enabled selection bias.

1: **function** BALANCABLE($\vec{u}, i, \tilde{r}$)
2: ▷ *Indices which allow balancing of at least* $\tilde{r} - 1$ *future ones*
3: **return** $\left\{ j \,\middle|\, \sum_{j'=1}^{j} \max\{0, 1 - \vec{u}[j'] + \min(\vec{u})\} \geq \tilde{r}, j \in \{1, \ldots, i-1\} \right\}$
4: **end function**

5: **function** GENERATE(N, r, r_1)
6: **init** $B \leftarrow 0 \in \mathbb{B}^{N \times r}$ ▷ Result matrix
7: **init** $\vec{u} \leftarrow 0 \in \mathbb{N}^r$ ▷ Bit usage count vector
8: **init** $\aleph^{\perp}$ ▷ Initialise root node
9: **for** $n \leftarrow 1$ **to** N **do** ▷ Iterate over all N samples
10: $\aleph \leftarrow \aleph^{\perp}, i \leftarrow r + 1$ ▷ Current node, start with root
11: **for** $\ell \leftarrow 0$ **to** $r_1 - 1$ **do** ▷ Trie level ℓ
12: $\tilde{r} \leftarrow r_1 - \ell$ ▷ Number of remaining one-bits $\tilde{r}$
13: $\mathfrak{s} \leftarrow \{j \mid \aleph^{\to j} > 0\}$ ▷ Indices with remaining paths
14: $\mathfrak{s} \leftarrow \mathfrak{s} \cap \{j \mid \vec{u}[j] = \min\{\vec{u}[j'] \mid j' \in \mathfrak{s}\}\}$ ▷ Balance bit usage
15: **if** BALANCABLE($\vec{u}, i, \tilde{r}$) $\cap\, \mathfrak{s} \neq \emptyset$ **then**
16: $\mathfrak{s} \leftarrow$ BALANCABLE($\vec{u}, i, \tilde{r}$) $\cap\, \mathfrak{s}$
17: **end if**
18: $j \leftarrow$ WEIGHTEDRANDOMSELECT($\{(j, \aleph^{\to j}) \mid j \in \mathfrak{s}\}$)
19: $B[n, j] \leftarrow 1$ ▷ Set output bit
20: $\vec{u}[j] \leftarrow \vec{u}[j] + 1$ ▷ Increment bit usage count
21: $\aleph^{\to j} \leftarrow \aleph^{\to j} - 1$ ▷ Decrement remaining path count
22: $\aleph \leftarrow \aleph[j], i \leftarrow j$ ▷ Go to child node j
23: **end for**
24: **end for**
25: **end function**

Algorithm 3.3: Algorithm for the generation of a block B of N unique, uncorrelated random vectors of length r, containing exactly r_1 ones with balanced bit allocation. Skipping lines 14–17 deactivates bit allocation balancing. Equation (3.28) describes how to initialise $\aleph^{\to j}$ upon first access. See text for description.



(a) Child node $j = 2$ is selected and inserted.

(b) The first r_1-combination $(4, 2, 1)$ has been generated, the cursor is reset to $\aleph^{\perp}$.

Figure 3.15: Continuation of the data generation algorithm example from Figure 3.13. In (a) child node 2 has been selected. Consequently, the only possible final node for this combination is $j = 1$. As indicated by the red usage count in (b), the index 4 cannot be chosen, as there are possible bits which have been used fewer times. The yellow usage count indicates an index with minimal usage count, but whose selection would not allow to balance the bit usage of larger indices. The remaining choice is to select bit index 5.

important detail is to bias the index selection by the number of possible paths continuing with j, $\aleph^{\to j}$. Failing to add the selection bias results in intracorrelation: if the first chosen bit index j is small, the next indices that can be selected must be smaller than j, causing small bit indices to appear in groups. This effect is shown in Figure 3.14: whereas there is a strong linear intracorrelation between lower (and as a result of balancing also higher) bit indices, the selection bias reduces the linear correlation coefficient between arbitrary bit indices to a small, uniform noise.

Note that in Figure 3.15(b) $(0, 0, 1, 1, 1)$ would be a possible next vector, but the algorithm can only generate $(0, 1, 1, 0, 1)$ or $(1, 0, 1, 0, 1)$ due to its greedy balancing strategy. We believe that fixing this issue is not worth the benefits: for reasonably large r the probability for such correlations is small.

Until now the algorithm generates N random and unique r_1-combinations. It does not fulfil the balancing condition from Equation (3.27). Balancing can be achieved by keeping track of the usage count of each bit index in a vector $\vec{u} \in \mathbb{N}^r$: whenever a bit index j is selected, the corresponding entry $(\vec{u})_j$ is incremented by one. We then restrict the set of currently selectable bit indices $\mathfrak{s}$ to those for which the usage count is minimal

$$\mathfrak{s}_{\min} = \mathfrak{s} \cap \{j \mid (\vec{u})_j = \min\{(\vec{u})_{j'} \mid j' \in \mathfrak{s}\}\} \quad \text{where } \mathfrak{s} = \{j \mid \aleph^{\to j} > 0\}\,. \tag{3.29}$$

As shown in Figure 3.15(b), this basic approach has to be refined: it may happen, that a bit index j has minimal usage. However, as we select the

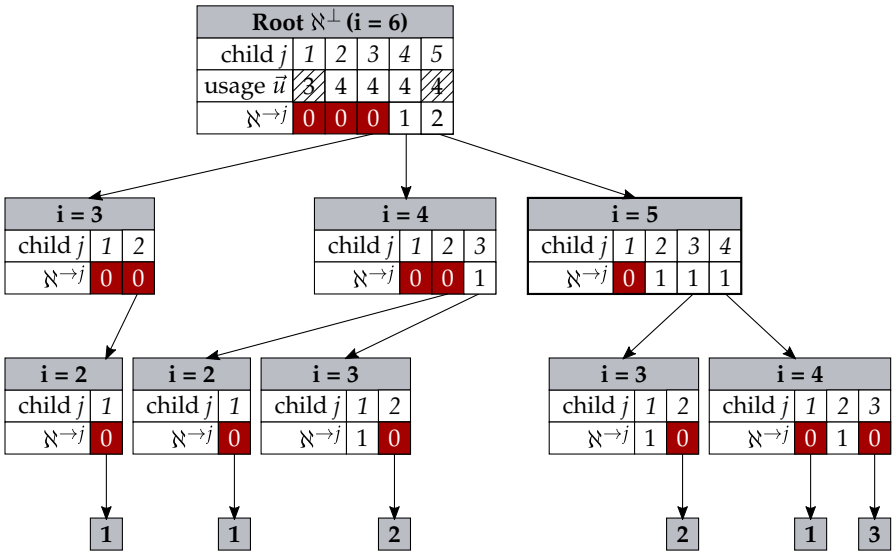


Figure 3.16: In certain situations the only possible choice introduces temporarily unbalanced vectors – either index 2, 3 or 4 must be chosen at the current node, although their bit usage is not minimal.

largest j in each step, there might be cases in which its selection would prevent a larger bit index to be balanced. We therefore calculate the bit indices $\mathfrak{s}_{\text{bal}}$ which allow the selection of at least $\tilde{r} = r_1 - \ell$ remaining ones with minimal usage

$$\mathfrak{s}_{\text{bal}} = \left\{ j \;\middle|\; \sum_{j'=1}^{j} \max\{0, 1 - (\vec{u})_{j'} + \min(\vec{u})\} \geq \tilde{r}, j \in \{1, \ldots, i-1\} \right\} . \tag{3.30}$$

The final selectable indices $\mathfrak{s}_{\text{sel}}$ are then given as

$$\mathfrak{s}_{\text{sel}} = \begin{cases} \mathfrak{s}_{\text{min}} \cap \mathfrak{s}_{\text{bal}} & \text{if } \mathfrak{s}_{\text{min}} \cap \mathfrak{s}_{\text{bal}} \neq \emptyset \\ \mathfrak{s}_{\text{min}} & \text{otherwise} \end{cases} . \tag{3.31}$$

Rationale for the second case is given in Figure 3.16: in intermediate steps it may be necessary to generate a temporarily unbalanced bit usage.

Algorithm 3.3 shows pseudo-code describing the method in its entirety. With $\mathcal{O}(N \cdot r_1 \cdot r)$ the runtime of the method linearly depends on N and r, which is worse than the previously presented $\mathcal{O}(N \cdot r_1)$ algorithm for non-balanced, non-unique samples.

3.4 CONCLUSION

This chapter outlined the design space of spiking BiNAM implementations: apart form the actual neuron parameters, the design space consists of the data parameters, the population size and the chosen input and output signal encoding (cf. Table 3.1). Furthermore, we

discussed the network testing procedure and possible evaluation measures, as well as different test sample generation schemes, for which we presented efficient algorithms.

Regarding the latter, experiments show that balanced input and output vectors $(\vec{x}_k, \vec{y}_k)$ without duplicates produce a more uniform activation of the synapses during experiments and achieve a higher possible storage capacity than randomly selected vectors.

With the material presented in this chapter we could implement a design space exploration pipeline which performs data generation, network construction, actual simulation on either a software integrator or one of the neuromorphic hardware systems, and evaluation according to the proposed measures.

However, such full network simulations are time-consuming and of limited feasibility for parameter optimisation with respect to one of the measures. Chapter 4 will thus approach the problem from a higher abstraction level and introduce estimations which predict the performance of a given neuron parameter set Φ for the network parameters listed above. We then move to the actual full network experiments on neuromorphic hardware in Chapter 5.

4 NEURON PARAMETER EVALUATION AND OPTIMISATION

A primary goal of this thesis is to explore the design space of the spiking BiNAM implementation presented in Chapter 3. In this chapter we summarise and categorise the design space parameters introduced in the preceding two chapters and discuss two strategies for design space exploration: full network and single neuron evaluation. Whereas the first strategy evaluates entire BiNAM networks according to memory performance measures, this chapter focuses on the second strategy, which examines how well a single neuron with parameters Φ matches the threshold and reset conditions introduced in Section 3.1.4.

Section 4.1 elaborates on single neuron and full network evaluations, and motivates single neuron evaluation as an approximation of full network evaluation. Section 4.2 details the implementation of the single neuron simulator, which is a building block of the individual evaluation measures discussed in Sections 4.3 to 4.5. A glimpse at the corresponding software toolchain is given in Section 4.6. This toolchain is then used to compare the measures regarding their accuracy and suitability for automated parameter optimisation in Section 4.7.

4.1 DESIGN SPACE EXPLORATION

This section compares two complementary strategies to design space exploration: full network exploration on the one hand, introduced in Chapter 3, and single neuron evaluation on the other hand, the primary topic of this chapter. Before we set out to discuss design space exploration, we need to clarify the notions of "design space" and "exploration". The first part of this section summarises the design space parameters described in Chapters 2 and 3, while the second and third part elaborate on the already mentioned exploration strategies. The fourth and final part discusses to what extent the design space dimensionality can be reduced and which constraints have to be imposed on the parameter combinations.

4.1.1 *On the terms "design space" and "exploration"*

The BiNAM design space can be partitioned into two disjunct sets of parameters: *system* and *neuron* parameters. System parameters specify the spike time encoding of input and output data, the amount of noise present on the platform, and the memory size and data characteristics. The neuron parameters Φ control the behaviour of the dynamical systems which constitute the network. Number and availability of neuron parameters depends on the chosen model. Here, we either

Detailed information on the system parameters can be found in the previous chapter 3, both the LIF and AdEx model and their parameters are presented in Section 2.3.

DESIGN SPACE OVERVIEW		
System parameters		
NETWORK	ENCODING	NOISE
Memory size m, n	Burst size s^{in}, s^{out}	False pos./neg. p_1, p_0
Number of "ones" c, d	Interspike interval Δt	Time noise σ_t, σ_t^{offs}
Sample count N	Sample interval T	Neuronal noise σ_ϕ □
Population size K		
Neuron parameters		
SYNAPSE	MEMBRANE (LIF)	ADEX
Weight w	Capacitance C_{m} ○	Adaptation a, b
Rev. potentials E_{e}, E_{i} ◇	Leak potential E_{L} ○	Time constant τ_{a}
Time constants τ_{e}, τ_{i} ◇	Threshold E_{Th} ○	Exp. threshold $E_{\text{Th}}^{\text{exp}}$
	Leak conductance g_{L}	Exp. slope Δ_{Th}
	Reset potential E_{reset}	
	Refractory period τ_{ref}	

Table 4.1: Overview of variables in the design space. ◇ Inhibitory synapses are not used in the BiNAM network, so the parameters E_{i} and τ_{i} can be ignored. ○ Superfluous degrees of freedom (Section 4.1.4). □ There is an individual noise parameter for each neuron parameter ϕ.

chose the linear integrate-and-fire (LIF) model or its extension, the adaptive exponential integrate-and-fire (AdEx) model. From a practical point of view, the most important distinction between neuron and system parameters is that system parameters are constant, external parameters, whereas the neuron parameters Φ must be tuned with respect to the system parameters, such that they fulfil the task specified by the memory parameters and encoding scheme, while constrained by the noise in their environment. Table 4.1 summarises the 16 neuron and 30 system parameters (14 plus 16 for neuronal noise).

At this point, we must distinguish two major applications of design space exploration. Sweeping over a static, manually defined subspace of the design space allows to compare the behaviour of the BiNAM on different platforms, which is potentially useful as a platform benchmark. The second application of design space exploration is parameter optimisation. Here, we try to find parameters, which optimise the memory with respect to one or multiple evaluation measures like those presented in Section 3.2. The optimisation process can be either manual, in which case an interactive visualisation of the design space is beneficial, or automated, in which case "smooth" gradients in the evaluation measures are favourable. Automated parameter optimisation facilitates the adaptation of a BiNAM implementation to new simulator platforms and topology parameters. In the remainder of this

section we focus on optimisation of a neuron parameter vector Φ with respect to constant system parameters. Optimisation of spiking neuron parameters is a problem commonly considered as hard to solve [Pri07].

4.1.2 *Full network evaluation*

Full network evaluation is the empirical approach to design space exploration. For each distinct parameter vector a network simulation is executed and the evaluation measures presented in Section 3.2 are calculated: storage capacity, latency, robustness and energy consumption. To visualise a two-dimensional sub-region of the parameter space, a network simulation can be executed for each point in a pre-defined two-dimensional grid. However, a major drawback of this methodology is the time required for network simulations. State-of-the-art neuromorphic hardware or fewer calculation points can not sufficiently alleviate this problem, because high setup and post-processing costs still apply. Naively sweeping over parameter space dimensions is off the table, as it is furthermore likely to waste vast portions of time in "uninteresting" parameter space regions, in which the neurons do not fulfil the threshold- and reset conditions required for an associative memory.

At the time of writing, setup and postprocessing times are in the order of seconds to minutes for considerably large networks simulated on the NM-PM1 and NM-MC1 hardware systems.

Calculating evaluation measures in such regions of the design space would lead to trivial results. Of course, a neuron which fires a single output spike whenever it receives an input spike – sketched in Figure 4.1 – would minimise the latency measure, but is useless in an actual associative memory. Classically, such degenerate solutions in multi-objective optimisation problems can be avoided by the introduction of a compound goal function which combines the aforementioned evaluation measures into a scalar optimality measure. Another means is Pareto optimisation, where parameter vector sets are searched, for which the variation of a single parameter dimension would degrade at least one of the performance measures. However, Pareto optimisation is less feasible for high-dimensional parameter spaces [Mie98].

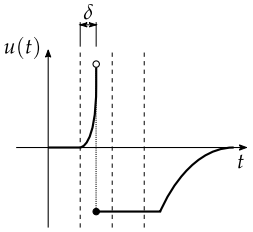
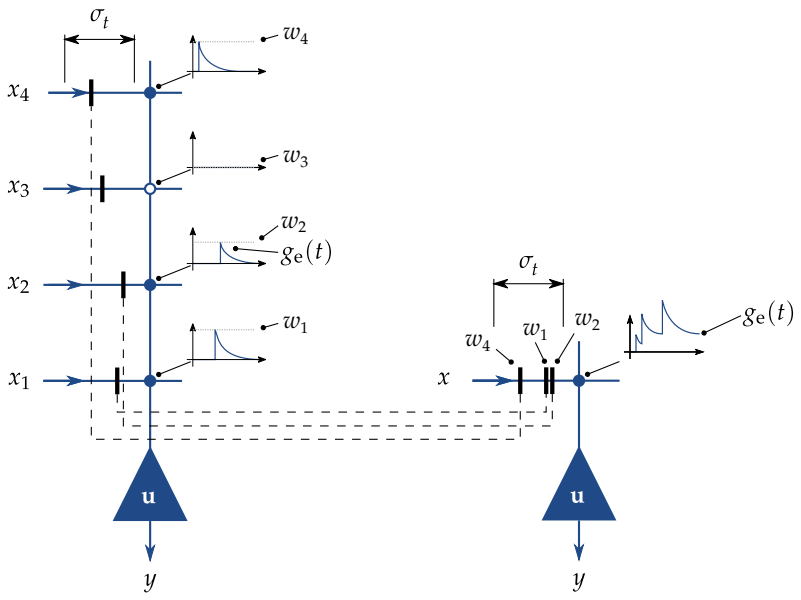


Figure 4.1: Sketch of $u(t)$ for a neuron with degenerate parameters. Issuing a single output spike immediately following the first input spike (dashed) and then staying in the refractory state minimises the latency δ.

4.1.3 *Single neuron evaluation*

The single neuron evaluation technique evades the perils of multi-objective optimisation and focuses solely on compound measures which quantify the "optimality" of neuron parameters. This approach is possible, as the BiNAM itself, and the spiking BiNAM implementation presented in Chapter 3 in particular, are highly homogeneous. Each individual neuron in the network is independently responsible for a single output component $(\vec{y})_i$, or, in case of neuron populations, a single signal in the output bundle. To provide an operational memory with a storage capacity close to the theoretical optimum, each neuron



(a) Individual inputs (b) Fused input

Figure 4.2: Sketch of the input spike train fusion model. (a) shows a single neuron receiving input spikes with timings drawn from a Gaussian distribution with standard deviation σ_t. The small diagrams depict the individual synapse conductivity over time. (b) shows an equivalent setup with a single input: the input spike trains arriving at non-zero weight synapses are fused into a single spike train with annotated weights.

in the network must equally adhere to the reset and threshold conditions postulated in Section 3.1.4. Regarding the neuron model, an input spike arriving at a synapse just increases the excitatory channel conductance $g_e(t)$ of the post-synaptic neuron by the synaptic weight w_i. The input spikes can be fused into an equivalent compound spike train, as long as they are annotated with the corresponding synaptic weight (Figure 4.2).

In the BiNAM implementation, all synapses share the same weight w. However, we do not need to restrict our model that far, as handling of different synapse weights also allows the handling of synaptic weight noise σ_w.

To summarise, given equal input, neurons in the network are supposed to exhibit the same behaviour, and the input itself can be represented by a single spike train t^{in} with annotated synaptic weights w^{in}. Correspondingly, design space exploration is possible by simulating a proxy neuron with parameters Φ, one excitatory synapse with adaptive weight and an according input spike train $(t^{\text{in}}, w^{\text{in}})$ which tests the neuron properties. Since the network solely consists of non-interacting copies of the proxy, its behavioural properties can be extrapolated to the entire network. In case the network is simulated deterministically by software, this reductionistic approach should accurately predict the behaviour of the network, allowing time-efficient and interactive design space exploration due to the vastly reduced complexity $\mathcal{O}(1)$ of single neuron simulation compared to the simulation of a full network in $\mathcal{O}(n)$.

However, by definition, the single neuron approach cannot account

for imprecisions on neuromorphic hardware. Apart from noise that might be introduced by analogue hardware on neuron parameters and synapse weights, both analogue and digital systems can produce non-uniform spike-time latencies and loose entire spike events due to limited network capacity. This kind of noise is likely to manifest when the data parameters m, n, c and d are increased. It is improbable that the simple Gaussian noise model presented in Section 3.1.2 correctly captures this behaviour. On an even more fundamental level, it is unclear whether the hardware system – apart from noise – realises a given parameter combination correctly. Thus, single neuron simulation can in principle not replace the execution of full networks on neuromorphic hardware. However, it allows to identify regions in the parameter space in which neurons potentially fulfil the requirements for neurons in an operational BiNAM spiking associative memory. Subsequently, a full network evaluation can be performed in these restricted regions.

4.1.4 *Parameter constraints and intra-dependencies*

With 16 dimensions, our parameter space does not lend itself to exhaustive exploration. However, by exploiting dependencies between parameters, constraints and redundancies, a smaller subspace of feasible parameters can be identified.

Invalid parameter combinations The first category of constraints in the AdEx model parameter space stems from the natural value range of some parameters. It should be obvious, that all time constants, capacitances and conductances must be larger than or equal zero. It must hold

$$C_\mathrm{m} > 0, g_\mathrm{L} > 0, \tau_\mathrm{e} > 0, \tau_\mathrm{i} > 0, \tau_\mathrm{a} > 0, \tau_\mathrm{ref} \geq 0, w \geq 0, a \geq 0\,. \tag{4.1}$$

The second category contains more debatable constraints. Perhaps the model might still produce sensible results if these constraints are violated, yet assumptions made in the following sections are no longer valid. The first set of constraints imposes an order on the potential parameters. For the AdEx model it should hold

$$E_\mathrm{Th} > E_\mathrm{e} > E_\mathrm{Th}^\mathrm{exp} > E_\mathrm{L} \geq E_\mathrm{i} \geq E_\mathrm{reset}\,, \tag{4.2}$$

for the LIF model, which does not possess the $E_\mathrm{Th}^\mathrm{exp}$ parameter, it should hold instead

$$E_\mathrm{e} > E_\mathrm{Th} > E_\mathrm{L} \geq E_\mathrm{i} \geq E_\mathrm{reset}\,. \tag{4.3}$$

The exponential slope Δ_Th and the adaptation current b are subject to the constraints

$$\Delta_\mathrm{Th} < E_\mathrm{Th}^\mathrm{exp} - E_\mathrm{L}, b \geq 0\,. \tag{4.4}$$

NOMINAL NM-PM1 ADEX PARAMETER RESTRICTIONS

Parameter	*Range*	*Parameter*	*Range*
C_m	1 nF	E_L	−125 mV to 45 mV
g_L	1.9 nS to 22.2 nS	E_e	−125 mV to 45 mV
τ_e	1 ms to 100 ms	E_i	−125 mV to 45 mV
τ_i	1 ms to 100 ms	E_{Th}	−125 mV to 45 mV
τ_a	20 ms to 780 ms	E_{spike}	−125 mV to 45 mV
τ_{ref}	0.16 ms to 10 ms	E_{reset}	−125 mV to 45 mV
a	0 nS to 10 nS	Δ_{Th}	0.4 mV to 3 mV
b	0 pA to 86 pA	w	0 µS to 0.3 µS (4 bit)

Table 4.2: Nominal AdEx parameter constraints of the NM-PM1 platform [Pet+14]. Only valid for a speedup of 10 000 and the specified C_m.

Violation of the first condition may result in unstable neuron behaviour; see Section 4.4.4 for a derivation of the condition. Similarly, negative adaptation currents $b < 0$ excite the neuron and can trigger a self-amplifying cascade of output spikes.

The parameter space of numerical simulators is limited by floating point precision, or, in the case of NM-MC1 the precision of the 16-bit integers in which the parameters are stored.

Hardware constraints Numerical spiking network simulators offer an almost unlimited parameter space. This is not true for analogue neuromorphic hardware systems such as Spikey and NM-PM1, where the parameters are limited by physical constraints. Unfortunately, as the mapping from model to hardware parameters is subject to calibration, selected membrane capacitance and speedup factor, the exact limitations vary. Table 4.2 shows the nominal parameter ranges for NM-PM1 at the speedup of 10 000 and a model membrane capacitance of 1 nF.

Superfluous degrees of freedom The LIF and AdEx neuron models possess three superfluous degrees of freedom (DoFs). The membrane capacitance C_m can be seen as a scaling factor for conductances and currents, the leak potential E_L solely offsets the membrane potentials and the threshold potential E_{Th} scales the membrane potential range. Transformation of the parameters into a normalised space eliminates C_m, E_L and E_{Th} while preserving the overall neuron behaviour.

The neuron simulator implemented for this thesis (Section 4.2) eliminates the parameters C_m and E_L, but not E_{Th}. This allows to keep voltage- and time-scaling and thus facilitates interpretation of intermediate values. In addition to DoF elimination, all divisions are factored out of the differential equations, which turns exponential decay time constants τ into decay rates λ. Table 4.3 shows the internal neuron parameters and the corresponding parameter space transformation. In the following we assume that these transformations are performed transparently in the simulator.

REDUCED DOF ADEX AND LIF PARAMETERS AND NEURON STATE			
Potentials			
	UNIT	DESCRIPTION	TRANSFORMATION
$E_{\mathrm{Th}}{}'$	[V]	Threshold potential	$E_{\mathrm{Th}} - E_{\mathrm{L}}$
$E_{\mathrm{Th}}^{\mathrm{exp}\prime}$	[V]	Spike potential	$E_{\mathrm{Th}}^{\mathrm{exp}} - E_{\mathrm{L}}$
$E_{\mathrm{reset}}{}'$	[V]	Reset potential	$E_{\mathrm{reset}} - E_{\mathrm{L}}$
$E_{\mathrm{e}}{}'$	[V]	Excitatory reversal potential	$E_{\mathrm{e}} - E_{\mathrm{L}}$
$E_{\mathrm{i}}{}'$	[V]	Inhibitory reversal potential	$E_{\mathrm{i}} - E_{\mathrm{L}}$
Decay rates			
	UNIT	DESCRIPTION	TRANSFORMATION
f_{L}	$[\mathrm{s}^{-1}]$	Membrane leak rate	$g_{\mathrm{L}}/C_{\mathrm{m}}$
λ_{a}	$[\mathrm{s}^{-1}]$	Adaptation decay rate	$1/\tau_{\mathrm{a}}$
λ_{e}	$[\mathrm{s}^{-1}]$	Excitatory channel decay rate	$1/\tau_{\mathrm{e}}$
λ_{i}	$[\mathrm{s}^{-1}]$	Inhibitory channel decay rate	$1/\tau_{\mathrm{i}}$
Other parameters			
	UNIT	DESCRIPTION	TRANSFORMATION
f_{w}	$[\mathrm{s}^{-1}]$	Synapse weight	w/C_{m}
f_{a}	$[\mathrm{s}^{-1}]$	Subthreshold adaptation	a/C_{m}
Δ_{b}	$[\mathrm{V\,s}^{-1}]$	Spike-triggered adaptation	b/C_{m}
τ_{ref}	[s]	Refractory period	τ_{ref}
Δ_{Th}	[V]	Spike slope factor	Δ_{Th}
State variables			
	UNIT	DESCRIPTION	TRANSFORMATION
$u'(t)$	[V]	Membrane potential	$u(t) - E_{\mathrm{L}}$
$\Delta_{\mathrm{a}}(t)$	$[\mathrm{V\,s}^{-1}]$	Adaptation current	$I_{\mathrm{a}}(t)/C_{\mathrm{m}}$
$f_{\mathrm{e}}(t)$	$[\mathrm{s}^{-1}]$	Excitatory channel frequency	$g_{\mathrm{e}}(t)/C_{\mathrm{m}}$
$f_{\mathrm{i}}(t)$	$[\mathrm{s}^{-1}]$	Inhibitory channel frequency	$g_{\mathrm{i}}(t)/C_{\mathrm{m}}$

Table 4.3: Overview of parameters and state variables in the AdEx model with reduced degrees of freedom (DoFs) as used as intermediate representation in the single neuron simulator.

4.2 SINGLE NEURON SIMULATION

In this section we discuss the implementation of a high-performance single neuron simulation for the AdEx model. This single neuron simulator is given an input spike stream t^{in} with annotated weights w^{in} and utilised as a building block in the various incarnations of single neuron evaluation measures presented in the subsequent sections 4.3 to 4.5. First, we outline the neuron simulation loop and challenges associated with non-differentiable state changes in the neuron models, followed by considerations regarding numerical integration specific to the AdEx model. In the last part we discuss selected numerical differential equation integrators and compare their performance.

4.2.1 *Neuron simulation loop*

As elaborated in the last section 4.1, the primary rationale for single neuron evaluation is to provide an efficient mechanism for design space exploration. While a single neuron could be simulated by any spiking neural network simulator, these programs carry a significant infrastructure and performance overhead for inter-neuron spike propagation: individual neuron simulations have to be synchronised after each time step to wait for input spikes generated by other parts of the network. In the single neuron use-case, the input spike train t^{in} is predefined. The simulator can run in a tight loop without expensive synchronisation [Mor+07]. A custom LIF and AdEx neuron model integrator conceived for this special application is thus beneficial from a time-performance point of view. Algorithmically, the implementation of a spiking neuron simulator involves numerical integration of the differential equations presented in Section 2.3. While generally the implementation of a differential equation integrator is straight-forward, a few minor hurdles have to be overcome for an efficient single neuron AdEx simulator.

An introduction to differential equation integrators is given in Section 4.2.3.

A system of differential equations f is called *autonomous* if it does not directly depend on t:

$$\mathrm{d}/\mathrm{d}t\ \vec{v}(t) = f\left(\vec{v}(t)\right) . \tag{4.5}$$

The evaluation of autonomous systems is potentially faster than that of equivalent non-autonomous systems $f\left(t, \vec{v}(t)\right)$. Since there are no time-dependent terms which have to be re-evaluated, this is especially true if the differential equation integrator performs multiple sub-timesteps. Unfortunately, the differential equations describing the LIF and AdEx model are non-autonomous: the refractory period mechanism (Equation (2.16)) and the input spike handling (Equation (2.10)) both depend on the simulation time t. Another challenge for numerical integration are the non-differentiable updates of the system state $\vec{v}$ for output spike generation, at the end of the refractory period, and at the arrival

given $t^{\text{in}}, w^{\text{in}}$ ▷ Input spike times and weights
given e, h, t_{end} ▷ Target error, step size and simulation end time
given $f(inRef, \vec{v})$ ▷ Neuron model specific differential equation
init $\vec{v} = (u, \Delta_{\text{a}}, f_{\text{e}}, f_{\text{i}})^\top$ ▷ Neuron state
init $t^{\text{out}} \leftarrow ()$ ▷ Empty output spike sequence
init $t \leftarrow 0, t_{\text{spike}} \leftarrow -\tau_{\text{ref}}$ ▷ Initialise simulation and last spike time
init $i_{\text{spike}} \leftarrow 1$ ▷ Current input spike index
while $t < t_{\text{end}}$ **do**
    $t_{\max} \leftarrow t_{\text{end}}$ ▷ Maximum integrator end time
    if $i_{\text{spike}} \leq |t^{\text{in}}|$ **then** ▷ Any unhandled input spike left?
        if $t \geq t^{\text{in}}[i_{\text{spike}}]$ **then** ▷ Handle input spikes
            if $w^{\text{in}}[i_{\text{spike}}] > 0$ **then** ▷ Adapt the synaptic channel rates
                $f_{\text{e}} \leftarrow f_{\text{e}} + w^{\text{in}}[i_{\text{spike}}]$
            else
                $f_{\text{i}} \leftarrow f_{\text{i}} - w^{\text{in}}[i_{\text{spike}}]$
            end if
            $i_{\text{spike}} \leftarrow i_{\text{spike}} + 1$ ▷ Next input spike
            continue ▷ Repeat in case of multiple input spikes
        end if
        $t_{\max} \leftarrow \min\{t_{\max}, t^{\text{in}}[i_{\text{spike}}] - t\}$ ▷ Stop at the next input spike
    end if
    $inRef \leftarrow (t - t_{\text{spike}}) < \tau_{\text{ref}}$ ▷ Test for refractory period
    if $inRef$ **then** ▷ Stop at the end of the refractory period
        $t_{\max} \leftarrow \min\{t_{\max}, \tau_{\text{ref}} + t_{\text{spike}}\}$
    end if
    $\vec{v}, h \leftarrow \textsc{Integrate}(f \circ inRef, \vec{v}, \min\{h, t_{\max} - t\}, t_{\max} - t, e)$
    if $u > E_{\text{Th}}$ **then**
        $t^{\text{out}} \leftarrow t^{\text{out}} \| (t)$ ▷ Append the output spike time to the result
        $u \leftarrow E_{\text{reset}}$ ▷ Reset membrane potential
        $t_{\text{spike}} \leftarrow t$ ▷ Start refractory period
        $\Delta_{\text{a}} \leftarrow \Delta_{\text{a}} + \Delta_{\text{b}}$ ▷ Spike-triggered adaptation
    end if
    $t \leftarrow t + h$ ▷ Advance t by the actually taken timestep h
end while
return t^{out} ▷ Return the output spike times

Algorithm 4.1: Basic single neuron simulator loop. Given an input spike time sequence t^{in} with annotated weights w^{in}, the algorithm calculates output spike times t^{out} of a single AdEx neuron. The system of differential equations is described in the functional f. $inRef$ specifies whether the neuron is in the refractory state, during which the differential $\dot{u}$ must be set to zero. Implementations of the INTEGRATE function are discussed in Section 4.2.3.

of input spikes. As the word suggests, non-differentiable behaviour is problematic for off-the-shelf differential equation integrator.

Fortunately, workarounds for these limitations exist. The refractory period is known in advance to end at $t_{\text{spike}} + \tau_{\text{ref}}$, where t_{spike} is the time of the last output spike. Furthermore, in the special case of single neuron simulation, the input spike times t^{in} are known in advance. Two of three non-differentiable state changes can be eliminated, and the system of differential equations transformed to an autonomous system,

if the integrator stops at these state changes boundaries. This can be accomplished by setting the maximum integrator timestep h to $t_{\max} - t$, where $t_{\max}$ is the time of the next input spike or end of the refractory period, whichever is earlier. As soon as the integrator stops at this boundary, the non-differentiable system state update is performed outside the integrator. The voltage-dependent reset-mechanism can also be performed outside the integrator. However, as demonstrated in Section 4.2.4, internal coupling of the neuron state variables in combination with temporary above-threshold membrane potentials causes imprecisions during integration.

More technical details of the simulator are discussed in Section 4.6.3.

Both, the transformation to an autonomous system and external non-differentiable updates, are employed in the neuron simulator implemented for this thesis, and are outlined in Algorithm 4.1.

4.2.2 *Numerical integration of the AdEx model*

The basic LIF model does not share these problems. An advanced model designed for efficient numerical integration is the MAT model [KTS09].

The AdEx model was not especially designed with suitability for numeric integration in mind. Use of the exponential as non-linearity poses two challenges worth discussing.

Threshold current limitation Spikes in the AdEx model are produced by a current I_{Th} which rises exponentially with the membrane potential u (Equation (2.21)):

$$I_{\mathrm{Th}} = g_{\mathrm{L}} \cdot \Delta_{\mathrm{Th}} \cdot \exp\left((u - E_{\mathrm{Th}}^{\mathrm{exp}}) \cdot {\Delta_{\mathrm{Th}}}^{-1} \right) \tag{4.6}$$

In a multi-step numerical differential equation integrator, naive integration of I_{Th} may cause undesired side effects. Due to the exponential, I_{Th} reaches extreme values for membrane potentials $u > E_{\mathrm{Th}}^{\mathrm{exp}}$. Such potentials may occur in a multi-step differential equation integrator, since the non-differentiable membrane potential reset on $u > E_{\mathrm{Th}}$ is only performed after multiple sub-steps. Due to an exponential avalanche inside the integrator, the membrane potential u and the coupled adaptation current I_{a} (Equation (2.20)) may overshoot and cause severe numerical instabilities. A solution to this problem is the limitation of I_{Th} to a reasonable maximum $I_{\mathrm{Th}}^{\max}$ which (under the assumption of no other current) safely allows to cross the entire membrane potential dynamic range $E_{\mathrm{dyn}} = E_{\mathrm{Th}} - E_{\mathrm{reset}}$ in one timestep h, but prevents an uncontrolled exponential avalanche. $I_{\mathrm{Th}}^{\max}$ is given as:

$$h \cdot \dot{u}(t) = h \cdot \frac{g_{\mathrm{L}} \cdot I_{\mathrm{Th}}^{\max}}{C_{\mathrm{m}}} \leq E_{\mathrm{dyn}} \Leftrightarrow I_{\mathrm{Th}}^{\max} \leq E_{\mathrm{dyn}} \cdot \frac{C_{\mathrm{m}}}{g_{\mathrm{L}} \cdot h}\,. \tag{4.7}$$

The actual threshold current used in the integrator $I_{\mathrm{Th}}{}'$ is then defined as $I_{\mathrm{Th}}{}' = \min\left\{ I_{\mathrm{Th}}^{\max}, I_{\mathrm{Th}} \right\}$.

Fast exponential function Profiling shows that more than 25% of the simulation time is spent in the evaluation of the exponential in Equation (4.6) (Table B.3). Fortunately, approximations of the exponential function exist, which exploit the bit-level layout of IEEE 754 floating point numbers to reduce the exponential to

The utilised exponential function approximation is provided in the "fastapprox"-library [Min12].

$$e^x = 2^{x'} = 2^a \cdot 2^b \text{ with } x' = \ln(2)^{-1} \cdot x, a = \lfloor x' \rfloor, b = x' - a\,, \quad (4.8)$$

with the integer 2^a being encoded as exponent, and the fractional 2^b approximated with a few multiplications. More details on exponential function approximation can be found in [Sch99]. The results of a profiling run with activated approximation is given in Table B.4. The total runtime of the simulator decreases by 13%, whereas there is no significant increase in the integration error (compare Tables B.5 and B.7). The benchmark is further discussed in Section 4.2.4.

4.2.3 *Differential equation integrators*

Neglecting any input, the state $\vec{v} \in \mathbb{R}^\alpha$ of a neuron at time t is given as

For the AdEx model with excitatory and inhibitory conductance based synapses, the dimensionality α of the state-vector $\vec{v}$ is $\alpha = 4$. The state-vector $\vec{v}$ consists of the components $\vec{v} = (u, \Delta_\mathrm{a}, f_\mathrm{e}, f_\mathrm{i})$.

$$\vec{v}(t) = \int_0^t f(\vec{v}(t))\,\mathrm{d}t + \vec{v}_0 \quad \text{with} \quad f(\vec{v}(t)) = \frac{\mathrm{d}\vec{v}(t)}{\mathrm{d}t} \quad (4.9)$$

where $\vec{v}_0 \in \mathbb{R}^\alpha$ denotes the initial state of the neuron, and the functional $f : \mathbb{R}^\alpha \longrightarrow \mathbb{R}^\alpha$ specifies a model-dependent autonomous system of differential equations. For most neuron models, the above integral cannot be solved in a closed form. Instead, a numerical differential equation integrator must be used, which approximates $\vec{v}(t)$ in discrete steps of length h. The approximation error E is defined as difference between the solution for $\vec{v}$ and numerical approximation $\vec{v}'$:

$$E(t) = \|\vec{v}(t) - \vec{v}'(t)\| \quad (4.10)$$

A differential equation integrator is conventionally referred to as n-th order, if $E(t) < \epsilon \cdot h^{n+1}$ for arbitrary, but constant ϵ [Pre+07a]. For efficient neuron simulation, a differential equation integrator with sensible ratio between error and computational effort should be selected.

In other words: the approximation error reduces exponentially with the order of the integrator; or the number of required steps reduces exponentially with the integrator order for a constant error. Apparently this explains why the first-order Euler's method is often shunned.

Constant step size integrators Constant step size integrators are the most basic form of differential equation integrators. Given the current neuron state $\vec{v} \in \mathbb{R}^\alpha$, the system of differential equations f and a step size h, the integrator returns an updated $\vec{v}'$:

$$\vec{v}'(t+h) = \textsc{Integrate}(f, \vec{v}(t), h) \quad (4.11)$$

Euler's method linearly follows the gradient described by the differential equations f given the current state $\vec{v}(t)$

$$\vec{v}'(t+h) = \vec{v}(t) + h \cdot f(\vec{v}(t))\,. \quad (4.12)$$

The family of Runge-Kutta integrators inserts i intermediate approximation steps to achieve a better approximation for non-linear $\vec{v}(t)$. The general form of the Runge-Kutta method for autonomous differential equations is given as [Sto+93]

$$\vec{v}'(t+h) = \vec{v}(t) + h \cdot \sum_{j=1}^{i} b_j \cdot \vec{k}_j \quad \vec{k}_j = f\left(\vec{v}(t) + h \cdot \sum_{\ell=1}^{j-1} a_{j\ell} \cdot \vec{k}_\ell\right). \quad (4.13)$$

Using the coefficients b_j and $a_{j\ell}$ in Table B.1 this general equation can be used to express a variety of differential equation integrators, including the first-order Euler's method, the second-order Midpoint method, and the fourth-order Runge-Kutta method.

Adaptive step size integrators Adaptive step size integrators dynamically control the step size h in such a way, that the approximation error $E(t)$ is kept at a small constant value e. This allows to focus computational effort on regions with disruptive changes, such as the exponential spike generation in the AdEx model, while quickly progressing past gently sloped regions, such as the membrane potential u slowly converging to E_L. The algorithmic interface for an adaptive step size integrator is expanded by a maximum step size h_max and the target error e. The actual step size h' is returned and must be passed to the next iteration

$$h', \vec{v}'(t+h') = \textsc{Integrate}(f, \vec{v}(t), h, h_\mathrm{max}, e) \quad (4.14)$$

The challenge lies in the efficient estimation of the error $E(t)$. Naively, two integration steps of size $h/2$ could be chained and compared to the result of a whole step h. However, this approach is rather wasteful, since a total of three integration steps have to be performed. The brilliant idea of the fifth-order Dormand-Prince integrator is to select the Runge-Kutta coefficients (Table B.2) in such a way, that the intermediate Runge-Kutta steps can be repurposed to estimate $E(t)$ [DP80].

The implementation in this thesis is adapted from the adaptive fifth-order Dormand-Prince integrator presented in Numerical Recipes [Pre+07a]. However, a few modifications have been made. First of all, terms solely required for non-autonomous differential equations are eradicated. Secondly, a simple proportional step size controller is used instead of a PI-controller. The most important change though concerns the way in which new step sizes h' are chosen depending on the estimated error $E(t)$. Instead of accounting for the fifth-order approximation error by taking the fifth root

$$h' = h \cdot \left(\frac{1}{E}\right)^{1/5}, \quad (4.15)$$

a simple linear scaling of h by $1/E$ is employed. Not only does the computation of the fifth root induce a major computational overhead, it also degrades the performance of the overall method – presumably h' is overestimated in the original version of the equation.

NEURON SIMULATOR BENCHMARK RESULTS					
Integrator		*LIF*		*AdEx*	
		t [ms]	Δu [mV]	t [ms]	Δu [mV]
Euler	$h = 1\,\mu s$	2071.354	0.223	2244.048	0.361
	$h = 10\,\mu s$	130.804	0.739	182.803	0.930
	$h = 100\,\mu s$	12.139	2.508	15.416	3.530
	$h = 1\,ms$	1.198	8.079	1.651	8.358
Midpoint	$h = 1\,\mu s$	2412.932	0.033	2994.206	0.306
	$h = 10\,\mu s$	165.374	0.948	264.261	0.887
	$h = 100\,\mu s$	13.282	3.156	23.706	3.554
	$h = 1\,ms$	1.482	9.820	2.579	**42.230**
Runge-Kutta	$h = 1\,\mu s$	3873.598	0.033	5045.481	0.294
	$h = 10\,\mu s$	314.017	0.948	460.079	0.856
	$h = 100\,\mu s$	32.832	3.155	47.700	3.932
	$h = 1\,ms$	3.024	9.817	4.560	**48.258**
Dormand-Prince	$e = 1\,\mu$	2286.730	0.033	2989.163	0.294
	$e = 10\,\mu$	572.327	0.336	735.503	0.285
	$e = 100\,\mu$	**54.232**	**1.330**	**74.478**	**0.323**
	$e = 1\,m$	7.367	4.200	10.839	0.437
	$e = 10\,m$	2.621	10.879	4.739	2.164
	$e = 100\,m$	3.019	13.645	4.205	3.610

Table 4.4: Neuron simulator benchmark results. The column Δu shows the RMSE for the membrane potential traces, the t column the total simulation time. Highlighted numbers are explicitly referred to in the text.

4.2.4 *Integrator benchmark*

For constant error E mathematical theory promises an exponentially smaller computational cost for higher order differential equation integrators. Yet, it is not clear whether the overhead of more complex methods pays off in a practical usage scenario. To this end, a benchmark comparing the different methods with varying precision is described in the following.

Method A single neuron receives 100 random input spike bursts which either contain four excitatory spikes, four excitatory spikes and two inhibitory spikes, or three excitatory spikes, each with Gaussian jitter of $\sigma_t = 1\,ms$ within a time window of $T = 100\,ms$. The corresponding synaptic weights are chosen in such a way, that only the first kind of bursts produces an output spike, totalling to an average of 33 output spikes over ten seconds of simulated time. The same spike train is fed into a neuron simulated with different integrator setups. The RMSE with nearest-neighbour interpolation between recorded neuron state trace and ground truth is used as a quality measure. The ground truth is produced by a fourth-order Runge-Kutta simulation with $h = 100\,ns$ step size. Table 4.4 summarises the runtime t and voltage component RMSE Δu. Results for the AdEx model were computed

"Excitatory spike" and "inhibitory spike" must be read as "input spike with annotated excitatory or inhibitory synaptic weight". The spikes themselves are of course neither excitatory nor inhibitory.

The fourth-order Runge-Kutta method serves as the ground truth because it can be implemented in just six lines of code (LoC), rendering any implementation error improbable – in contrast to the 135 LoC adaptive Dormand-Prince integrator.

with enabled exponential function approximation. See Appendix B.3 for more detailed results.

Results As expected, the simulation time reduces for larger time steps h and target errors e, while the resulting error rises. Furthermore, the AdEx model is computationally more intensive than the LIF model. However, the results clearly lack the theoretically predicted decrease in error for higher order integrators. On the contrary, the error either stays approximately the same or increases for higher-order integrators. Astoundingly large errors are produced for a $h = 1\,\text{ms}$ time step and the AdEx model for the Midpoint and Runge-Kutta integrators. The adaptive step size Dormand-Prince integrator performs better in conjunction with the more complex AdEx model, than with the simpler LIF model. It does not show any severe increases in error.

Discussion The large errors produced by higher-order integrators for the AdEx model can be accounted to the problem predicted in Section 4.2.1. While a neuron simulated with Euler's method just resets when reaching the exponential spike generation mechanism, taking multiple sub-steps during a phase of exponential membrane potential growth allows to couple large membrane potentials into the adaptation current a (Table B.5), which in return influences the membrane potential after the reset. However, this does not explain why for LIF neurons the higher-order constant step size integrators are scarcely better than Euler's method. On a brighter note, the results clearly show that the time spent for implementing an adaptive step size integrator was worthwhile. For $e = 100 \cdot 10^{-6}$ and the AdEx model, a competitively small error of 0.3 mV can be reached ten to one hundred times faster. Although the advantage is smaller for the LIF model, the adaptive step size integrator is still about six to three times faster than the Midpoint and Runge-Kutta methods, and as fast as Euler's method. For sensible error values, the adaptive step size Dormand-Prince integrator is not worse performance-wise than constant step size integrators, and clearly outperforms them when used in conjunction with the AdEx model. Therefore, we select this integrator as basis for neuron simulation.

4.3 APPROACH 1: SPIKE TRAIN

In the previous sections we established the design space and presented a blueprint for a single neuron simulator. In the following we build upon these concepts and describe three single neuron evaluation measures, which assign abstract optimality values $\mathcal{P}$ to a parameter vector Φ. This section describes the so called "spike train" measure. We elaborate on the underlying idea, then describe the parameters of the so called "group descriptor" and finally discuss the equations describing the measure itself.

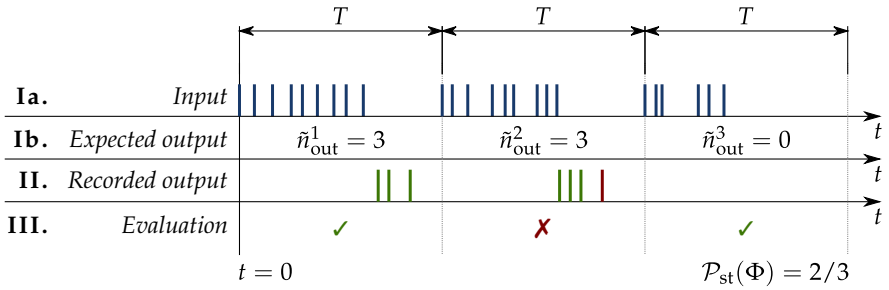


Figure 4.3: Example of a spike train evaluation. Bold vertical lines represent single input/output spikes. Row *Ia* shows the input spike train, separated into $n_g = 3$ compartments generated according to the group descriptors shown in Table 4.6 on the next page, a burst size of three and a group length T. Row *Ib* shows the number of expected output spikes for each experiment group. Row *II* depicts an examplatory result from a neuron. Accordance of the actual result with the actual result is illustrated in row *III*.

4.3.1 *Concept*

The spike train measure is a flexible, empirical approach to single neuron evaluation. It allows to test whether a neuron with parameters Φ responds as expected to certain inputs. As shown in Figure 4.3, the measure is separated into three stages. First, an input spike train, separated into n_g experiment groups of length T, is constructed. The assembly of each group follows a group descriptor, which is randomly chosen from a descriptor pool. The descriptor specifies the expected output spike count and the number of excitatory and inhibitory input bursts. In the second step the input is fed into a single neuron simulation with parameters Φ. In the third step the recorded output spike train is evaluated with respect to the expected spike count for each compartment, resulting in an estimated probability $\mathcal{P}_{\text{st}}$ of the neuron to fulfil the behaviour encoded in the group descriptors.

Generally, the measure can be used to examine the functionality of a single neuron in an arbitrary feed-forward network. With an appropriate pool of descriptors, it can be configured to test the BiNAM threshold condition, and implicitly the reset condition (Section 3.1.4). For large n_g, it serves as a reliable prediction of network behaviour and provides ground-truth data for the "single group" methods introduced in Sections 4.4 and 4.5.

4.3.2 *Descriptor and input spike train generation*

The compartments of the randomly generated input spike train t^{in} with annotated weights w^{in} are generated according to a pool of experiment group descriptors. The group descriptors specify the number of excitatory and inhibitory input bursts n_E and n_I, as well as the number of expected output spikes $\tilde{n}^{\text{out}}$. Individual input bursts are generated

It is important to distinguish the number of spikes and bursts. The expected output $\tilde{n}^{\text{out}}$ is measured in spikes, the input in bursts (each counting s^{in} spikes).

SPIKE TRAIN MEASURE META-PARAMETERS	
Global parameters	
n_g	Total number of experiment groups in the input spike train. Due to the random sampling performed in the method, a larger number of experiment groups results in a more accurate the result. Unless noted differently n_g is chosen as 100 in this thesis.
Experiment group descriptor parameters	
n_E	Number of excitatory input bursts.
n_I	Number of inhibitory input bursts. Inhibitory input bursts could be used in conjunction with more sophisticated networks.
$\tilde{n}^{out}$	Number of expected output spikes.
w_E	Weight factor of the excitatory input bursts, allows to simulate a larger synaptic weight for the excitatory synapse. Usually set to 1.0.
w_I	Weight factor of the inhibitory input bursts. Set to 1.0.

Table 4.5: Full list of meta-parameters in the "spike train" evaluation method.

according to the data encoding parameters in Section 3.1.2 and Table 3.1. Note that the input bursts in a spike train compartment are not time-sequential – they are fused, emulating the coincidental arrival of single input bursts at n_E excitatory and n_I inhibitory synapses of a neuron (Figure 4.2 and Section 4.1.3). Time offsets of single spikes are solely controlled by the data encoding parameters σ_t, σ_t^{offs} and Δt. The annotated weights in w^{in} are selected according to the specified synapse weight w with superimposed Gaussian noise according to the noise parameter σ_w. The group descriptor parameters w_E and w_I allow to rescale the weights for individual groups. The experiment group descriptors are summarised in Table 4.5.

Embedding the BiNAM threshold condition into the spike train measure framework requires two experiment group descriptors, one with $n_E = c \cdot K$ and an expected output spike count of $\tilde{n}^{out} = s^{out}$, and one with $n_E = (c-1) \cdot K$ and $\tilde{n}^{out} = 0$. An example is given in Table 4.6. As the n_g experiments are randomly sampled from these two descriptors and temporarily multiplexed with an interval T into a single input spike train, the reset condition is implicitly tested – if it was not fulfilled, the individual experiments would influence each other and the neuron would not produce the expected result.

n_E	n_I	$\tilde{n}^{out}$
2	*0*	*0*
3	*0*	*1*

Table 4.6: Example of group descriptors testing the threshold behaviour for $c = 3$, $K = 1$ and $s^{out} = 1$.

4.3.3 *Evaluation*

Given the input (t^{in}, w^{in}), the behaviour of a single neuron with parameters Φ is simulated and the output spike train t^{out} is recorded. Let $\tilde{n}_i^{out}$ denote the expected number of output spikes for group i and

$$n_i^{out} = |\{t_j^{out} \mid T \cdot (i-1) \leq t_j^{out} < T \cdot i\}| \tag{4.16}$$

denote the number of actual output spikes in the i-th spike train compartment. The evaluation measure $\mathcal{P}_{\text{st}}$ is then defined as ratio of successful experiments to the total number of experiments:

$$\mathcal{P}^i_{\text{st}}(\Phi) = \begin{cases} 1 & \text{if } \tilde{n}^{\text{out}}_i = n^{\text{out}}_i(\Phi) \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

$$\mathcal{P}_{\text{st}}(\Phi) = \frac{1}{n_{\text{g}}} \cdot \sum_{i=1}^{n_{\text{g}}} \mathcal{P}^i_{\text{st}}(\Phi) \tag{4.18}$$

For large n_{g} the target function $\mathcal{P}_{\text{st}}(\Phi)$ can be interpreted as the probability of the neuron with parameters Φ to exhibit the behaviour described in the experiment group descriptor pool.

4.4 APPROACH 2: SINGLE GROUP, SINGLE OUTPUT SPIKE

The just presented spike train measure $\mathcal{P}_{\text{st}}$ has major drawbacks: $\mathcal{P}_{\text{st}}(n_{\text{g}})$ is a discrete step function, which hinders automated, gradient based optimisation, and depending on the magnitude of the noise parameters, the number of randomly generated experiment groups n_{g} must be rather large to produce stable and exact output values. This potentially results in long simulation times, which mitigates the promised benefit of single neuron evaluations.

This section presents the "single group, single output spike" (SGSO) measure, which performs a limited number of independent experiments and calculates a smooth compound measure $\mathcal{P}_{\text{sgso}}$ for the special case that the number of output spikes encoding a "one" is exactly one. In the following, we discuss the general idea, describe the input spike trains used in the experiments, the equations for $\mathcal{P}_{\text{sgso}}$, and finally focus on the notion of the "effective threshold potential" $E^{\text{eff}}_{\text{Th}}$.

The "single group" part of the name stems from each independent simulation run to correspond to an experiment with a single group in the spike train measure.

4.4.1 *Concept*

The central idea of the SGSO measure is to analyse the distance between the maximum neuron membrane potential u_{max} and the effective threshold potential $E^{\text{eff}}_{\text{Th}}$. The latter is defined as the membrane potential that has to be surpassed for the neuron to inevitably generate an output spike. For single output spikes, the BiNAM threshold condition is fulfilled if u_{max} reaches $E^{\text{eff}}_{\text{Th}}$ for n_1 input spikes, but stays below the threshold for n_0 input spikes (Equation (3.7)). In either case, there should be a large margin between u_{max} and $E^{\text{eff}}_{\text{Th}}$ to increase robustness. The limitation to a single output spike is enforced by repeating the experiment for n_1 input spikes with $u(0) = E_{\text{reset}}$ and $t_{\text{spike}} = 0$. This puts the neuron into the refractory period and mimics the neuronal state following an output spike. Given these initial conditions, the neuron should not reach the threshold. The BiNAM reset condition is tested by capturing the neuron state in all three experiments at

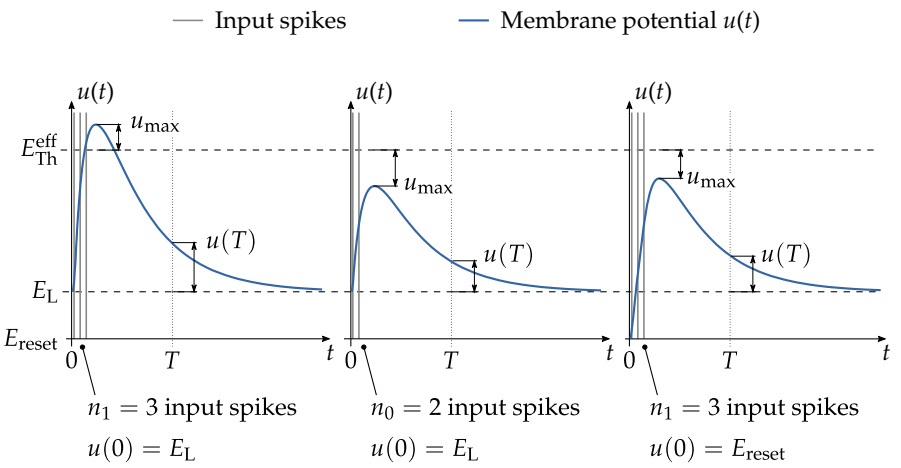


Figure 4.4: Conceptual overview of the single group, single output spike measure. Three independent experiments are conducted, in which a neuron with deactivated spiking mechanism is either presented n_0 or n_1 input spikes. The degree to which the neuron fulfils the threshold condition is determined by comparing the maximum membrane potential $u_{\max}$ to the effective threshold potential $E_{\mathrm{Th}}^{\mathrm{eff}}$. The neuron state at time T determines how well it fulfils the reset condition.

the time T and comparing the result to the initial neuron state. The method is sketched in Figure 4.4.

Deactivation of the spiking mechanism is the actual reason why the measure can only handle a single output spike. Expansions of the measure allowing multiple output spikes by running follow-up experiments which restarted the simulation at the threshold-crossing point did not result in a satisfactorily smooth measure.

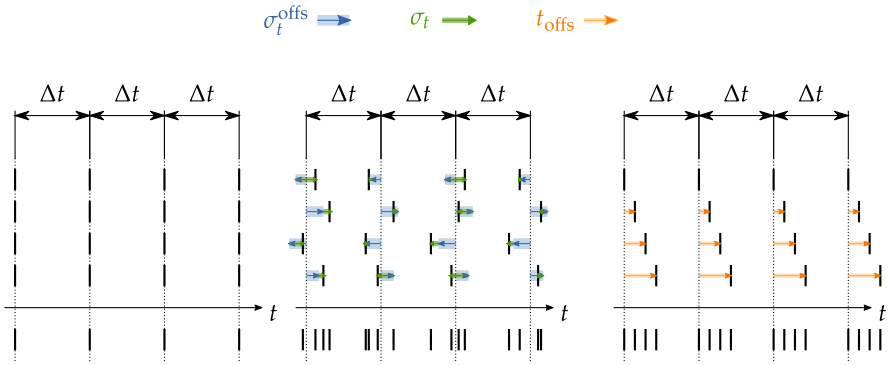
A major challenge of this methodology is that – by definition – the neuron issues an output spike as soon as $E_{\mathrm{Th}}^{\mathrm{eff}}$ is reached. As a result, it would hold $u_{\max} = E_{\mathrm{Th}}$. This impedes a quantification of *how much* the neuron surpasses the threshold, which in return hinders the goal to optimise for a large margin and to provide a smooth evaluation measure. The pragmatic solution to the problem is to simply deactivate the neuron spiking mechanism. In the context of the LIF model, the neuron must not reset once E_{Th} is reached. For the AdEx model, the exponential spike generation current must furthermore be limited to

$$I_{\mathrm{Th}}{}'(u) = \min\{I_{\mathrm{Th}}(u), I_{\mathrm{Th}}(E_{\mathrm{Th}}^{\mathrm{eff}})\}\,. \tag{4.19}$$

This prevents the membrane potential from rising to infinity but does not obstruct neuron dynamics below the threshold.

4.4.2 *Deterministic input spike train generation*

A further distinguishing goal of the SGSO measure is determinism. Until now, input spike train generation is based on a stochastic model with noise parameters σ_t and σ_t^{offs}. As shown in Figure 4.5(a), determinism could be enforced by setting the spike time noise parameters σ_t and σ_t^{offs} to zero. Statistically, such a spike train would correspond to averaging the spike times over an infinite number of randomly generated spike trains. However, this does not account for the σ_t- and σ_t^{offs}-induced spread of spike times encountered when fusing multiple



(a) No noise (b) Random bursts (c) Equidistant offset

Figure 4.5: Comparison of single neuron simulation spike train generation methods. The three sketches show four bursts each (top) which are fused into a single spike train for single neuron simulation (bottom). (a) shows four bursts generated without noise and an interspike interval Δt. The i-th spike of each burst is at exactly the same time. (b) shows bursts randomly generated according to the parameters σ_t and σ_t^{offs} as specified in Section 3.1.2. The resulting spike packets of the fused spike trains show a certain spread. (c) deterministically emulates the spread of the spike packets by offsetting each burst by a multiple of t_{offs}.

random bursts in a single spike train – see Figure 4.5(b). This spread can be emulated by adding a multiple of a constant offset t_{offs} to each burst in the input spike train. The offset is chosen as

$$t_{\text{offs}} = \frac{2 \cdot (\sigma_t + \sigma_t^{\text{offs}})}{n_{\text{bursts}}}, \tag{4.20}$$

where the denominator n_{bursts} is the number of input bursts (either c or $c-1$), and the numerator $2 \cdot (\sigma_t + \sigma_t^{\text{offs}})$ the width of the 68.2%-quantile of the Gaussian distribution resulting from additive superposition of the Gaussian distributions accounting for σ_t and σ_t^{offs}. Figure 4.5(c) sketches the effect of the equidistant offset.

4.4.3 *Evaluation measure*

With idea and input generation laid out, we proceed to the equations specifying $\mathcal{P}_{\text{sgso}}$. Let $u^1_{\max}$, $u^0_{\max}$ denote the maximum membrane potentials encountered in a single neuron simulation for c and $c-1$ input bursts respectively. The third potential $u^{\text{reset}}_{\max}$ denotes the maximum membrane potential for c input bursts with the neuron starting in its refractory state. Let $\mathrm{L}(x \mid x_0)$ denote a heavy-tailed sigmoid function

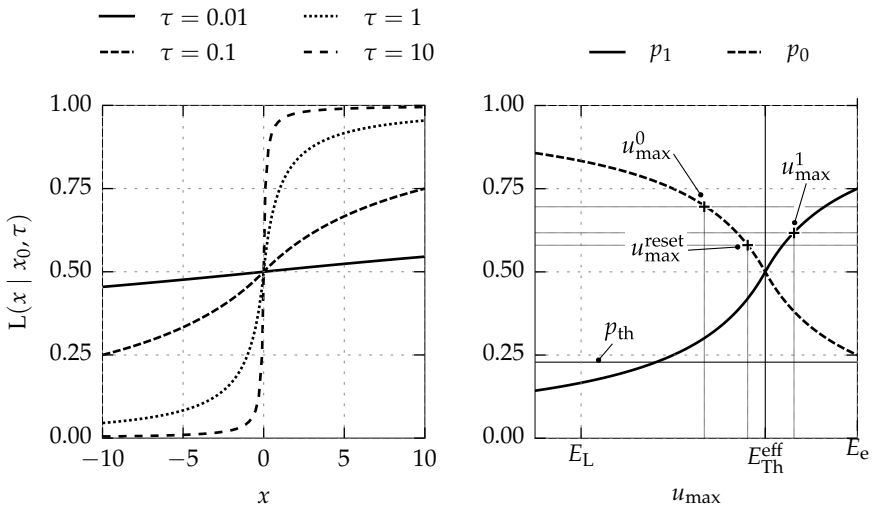
$$\mathrm{L}(x \mid x_0) = \frac{1}{2} \cdot \left(1 + \frac{\tau \cdot (x - x_0)}{1 + \tau \cdot |(x - x_0)|}\right). \tag{4.21}$$

An important property of $\mathrm{L}(x \mid x_0)$ is its non-exponential falloff, causing the sigmoid to saturate slowly. Goal of this choice is to facilitate parameter optimisation by providing a distinct gradient. Example

The free parameter τ in Equation (4.21) controls the slope of sigmoid function and consequently the "softness" of p_{th}. τ is chosen according to

$$\tau = \frac{1 - 2 \cdot \Delta p}{2 \cdot \Delta u \cdot \Delta p},$$

where Δp is the value of L at $-\Delta u$. Here these two values are $\Delta u = 2 \cdot 10^{-3}$ and $\Delta p = 0.2$.



(a) Heavy-tailed sigmoids for varying τ (b) Illustration for the calculation of p_{th}

Figure 4.6: Examples of heavy-tailed sigmoids and the threshold evaluation. (a) shows examples of the heavy-tailed sigmoid function in Equation (4.21) with $x_0 = 0$. In contrast to other sigmoid functions, the heavy-tailed sigmoid shows no exponential falloff. (b) visualises the calculation of p_{th} in Equation (4.22) for given u^1_{max}, u^0_{max} and $u^{\text{reset}}_{\text{max}}$. The measure p_{th} results from the multiplication of the values read off the sigmoids.

plots of $\mathrm{L}(x \mid x_0)$ are shown in Figure 4.6(a). The first ingredient to the evaluation measure, p_{th}, describes accordance with the threshold condition

$$p_{\text{th}} = \mathrm{L}(u^1_{\text{max}} \mid E^{\text{eff}}_{\text{Th}}) \cdot (1 - \mathrm{L}(u^0_{\text{max}} \mid E^{\text{eff}}_{\text{Th}})) \cdot (1 - \mathrm{L}(u^{\text{reset}}_{\text{max}} \mid E^{\text{eff}}_{\text{Th}})). \quad (4.22)$$

Figure 4.6(b) sketches the individual factors of the above equation for fixed potentials. Large u^1_{max}, and small u^0_{max} and $u^{\text{reset}}_{\text{max}}$ result in a high valuation in the compound measure p_{th}.

The final ingredient to the measure is an estimation of how well the reset condition is fulfilled. Let $\vec{v}^1_T$, $\vec{v}^0_T$ and $\vec{v}^{\text{reset}}_T$ denote neuron state vector of the corresponding simulation runs at time T, and $\vec{v}_0$ the initial neuron state. Accordance with the reset condition p_{reset} is measured as

$$p_{\text{reset}} = \exp\left(-\|\vec{v}^1_T - \vec{v}_0\| - \|\vec{v}^0_T - \vec{v}_0\| - \|\vec{v}^{\text{reset}}_T - \vec{v}_0\|)\right). \quad (4.23)$$

The norm $\|\cdot\|$ should be chosen such that individual vector dimensions are rescaled to similar value ranges. The final evaluation measure $\mathcal{P}_{\text{sgso}}(\Phi)$ for a parameter vector Φ is defined as

$$\mathcal{P}_{\text{sgso}}(\Phi) = p_{\text{th}}(\Phi) \cdot p_{\text{reset}}(\Phi). \quad (4.24)$$

4.4.4 *Effective threshold potential*

The effective threshold potential $E_{\text{Th}}^{\text{eff}}$ is defined as the membrane potential at which a neuron inevitably produces an output spike (Section 4.4.1). For the LIF model, it trivially holds $E_{\text{Th}}^{\text{eff}} = E_{\text{Th}}$. For the AdEx measure, $E_{\text{Th}}^{\text{eff}}$ corresponds to the unstable stationary point in the membrane potential bifurcation analysis (Figure 2.12(c)). Consider the exponential current I_{Th} in the AdEx model (Equation (2.22))

$$I_{\text{Th}} = g_{\text{L}} \cdot \Delta_{\text{Th}} \cdot \exp\left((u(t) - E_{\text{Th}}^{\text{exp}}) \cdot {\Delta_{\text{Th}}}^{-1} \right) . \tag{4.25}$$

For an absence of inhibitory currents ($I_{\text{a}}(t) \leq 0$ and $g_{\text{i}}(t) = 0$), and a leak potential $E_{\text{L}} = 0$, $E_{\text{Th}}^{\text{eff}}$ is given as the membrane potential x at which leak and threshold current cancel each other out

As discussed in Section 4.1.4, the leak potential is only an offset to all membrane potentials.

$$g_{\text{L}} \cdot x = g_{\text{L}} \cdot \Delta_{\text{Th}} \cdot \exp\left((x - E_{\text{Th}}^{\text{exp}}) \cdot {\Delta_{\text{Th}}}^{-1} \right) . \tag{4.26}$$

We now discuss two ways in which the equation can be solved for x.

Lambert W function Rearranging Equation (4.26) yields

The Lambert W function is defined as $W(z) = X(x) \Leftrightarrow z = X(x) \cdot \exp(X(x))$ with a real-valued result if $z > -\frac{1}{e}$ holds.

$$\begin{aligned} \Delta_{\text{Th}} &= x \cdot \exp\left(-(x - E_{\text{Th}}^{\text{exp}}) \cdot {\Delta_{\text{Th}}}^{-1} \right) \\ \Leftrightarrow -\exp\left(-E_{\text{Th}}^{\text{exp}} \cdot {\Delta_{\text{Th}}}^{-1} \right) &= -x \cdot {\Delta_{\text{Th}}}^{-1} \cdot \exp\left(-x \cdot {\Delta_{\text{Th}}}^{-1} \right) . \end{aligned} \tag{4.27}$$

This equation can be solved with the Lambert W function [Cor+96]

$$E_{\text{Th}}^{\text{eff}} = x = -\Delta_{\text{Th}} \cdot W\left(-\exp\left(-E_{\text{Th}}^{\text{exp}} \cdot {\Delta_{\text{Th}}}^{-1} \right) \right) . \tag{4.28}$$

$W(z)$ possesses a real-valued solution for $z > -\frac{1}{e}$. It must hold

$$\exp\left(-E_{\text{Th}}^{\text{eff}} \cdot {\Delta_{\text{Th}}}^{-1} \right) < \exp(-1) \Leftrightarrow E_{\text{Th}}^{\text{eff}} > \Delta_{\text{Th}} . \tag{4.29}$$

For $E_{\text{Th}}^{\text{eff}} \leq \Delta_{\text{Th}}$ the threshold current I_{Th} would always be larger than the leak current. The neuron would be in an unstable state in which it repeatedly generates output spikes.

Iterative calculation For a practical implementation of the calculation of $E_{\text{Th}}^{\text{eff}}$ without the Lambert W function, Equation (4.26) can be solved iteratively using Newton's method. However, it is important to apply the logarithm to the equation

Newton's method is likely to diverge if the logarithm is not applied, as the derivative still contains an exponential function.

$$f(x) = 0 = \log(\Delta_{\text{Th}}) + (x - E_{\text{Th}}) \cdot {\Delta_{\text{Th}}}^{-1} - \log(x) . \tag{4.30}$$

The iterative solution is given as

$$\begin{aligned} x_0 &= E_{\text{Th}} , \\ x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{\log(\Delta_{\text{Th}}) + (x - E_{\text{Th}}) \cdot {\Delta_{\text{Th}}}^{-1} - \log(x)}{{\Delta_{\text{Th}}}^{-1} - x^{-1}} , \end{aligned} \tag{4.31}$$

where $E_{\text{Th}} > 0$ and $\Delta_{\text{Th}} > 0$. Under these circumstances the formula usually converges to a solution with an accuracy of 1 nV in three to four steps.

4.5 APPROACH 3: SINGLE GROUP, MULTIPLE OUTPUT SPIKES

In contrast to the spike train measure, the SGSO measure produces a smooth, non-discrete output. Yet, the former accurately models the environment of a single neuron in a BiNAM network which receives a new sample in an interval T. Furthermore, all properties of the neuron are tested, including the spiking mechanism, refractory period and the adaptation mechanism, and there is no limitation regarding the number of output spikes.

Combining the advantageous properties of the previous two measures is the goal of the "single group, multiple output spikes" measure (SGMO). It aims at respecting all neuron model characteristics, allowing for an arbitrary expected output spike number, while still being computationally inexpensive and guaranteeing a smooth evaluation measure. Section 4.5.1 presents the general idea on how to tackle the lofty goals mentioned above. It furthermore provides the defining equations of the measure, which assume the existence of a *fractional spike count* q^{out}. An efficient algorithm for the calculation of the said q^{out} is covered in the second part, Section 4.5.2.

4.5.1 *General idea*

Again, the overall goal of the measure is to facilitate neuron parameter optimisation with respect to the threshold and reset conditions (Section 3.1.4). Optimisation of the threshold condition can be formulated as minimisation of the error E in the following framework

$$E(\Phi) = |\tilde{n}^{\text{out}} - n^{\text{out}}(\Phi, t_1^{\text{in}})| + n^{\text{out}}(\Phi, t_0^{\text{in}}) \,, \tag{4.32}$$

where t_1^{in} and t_0^{in} are deterministic input spike trains (Section 4.4.2) for which the neuron is expected to produce $\tilde{n}^{\text{out}}$ and zero output spikes respectively. An eminent disadvantage of the above equation are the discrete spike counts n^{out}, which cause the error function $E(\Phi)$ to take the form of a step function.

A fractional output spike count $q^{\text{out}} = 3.5$ should for example be interpreted as "the neuron outputs three spikes, and is already half way to generating a fourth spike".

The crucial idea is to replace discrete spike counts n^{out} with fractional spike counts $q^{\text{out}} \in \mathbb{R}^+$, which encode both the integral number of output spikes n^{out}, as well as the estimated likelihood of another output spike p^{out}. Assuming the existence of such a measure q^{out}, the error equation can be reformulated as

$$E(\Phi) = |o + \tilde{n}^{\text{out}} - q^{\text{out}}(\Phi, t_1^{\text{in}})| + q^{\text{out}}(\Phi, t_0^{\text{in}}) \,, \tag{4.33}$$

where the offset o should theoretically be chosen as $o = 1/2$ to maximise the margin between two discrete spike count steps: the neuron should not be close to issuing $\tilde{n}^{\text{out}} - 1$ spikes, but also not too close to issuing $\tilde{n}^{\text{out}} + 1$ spikes. In case q^{out} is not linear with respect to the variation of a single parameter dimension, another choice of o might yield a more robust behaviour. The value chosen here is $o = 0.3$.

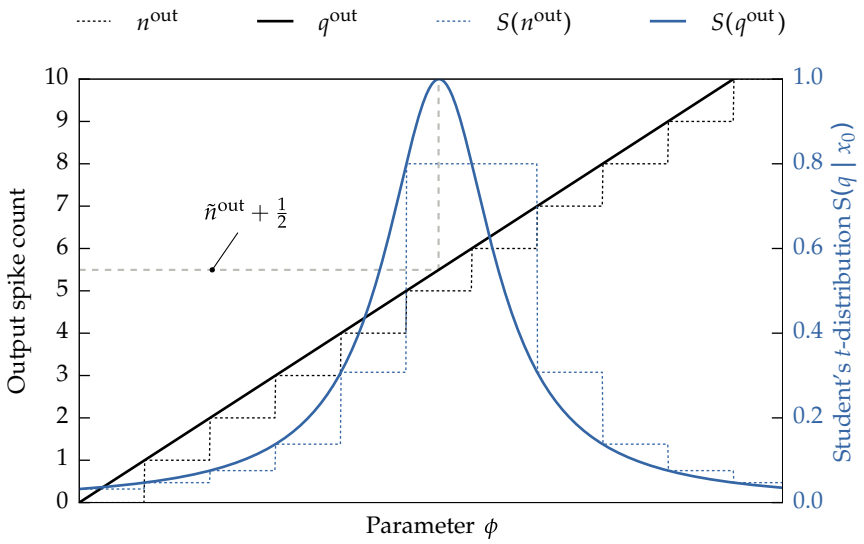


Figure 4.7: Idealised sketch of the SGMO measure. The output spike count $n^{\text{out}}(\phi)$ over a neuron parameter ϕ is smoothed to a fractional spike count measure $q^{\text{out}}(\phi)$ (here in its ideal linear form). The fractional spike count is converted to a bell-shaped function (an unnormalised Student's t-distribution) with zero to one value range, which indicates how well the current spike count matches the target spike count $\tilde{n}^{\text{out}}$.

The additive error value $E(\Phi)$ is rewritten as pseudo-probabilistic value $p_{\text{th}}(\Phi)$ consisting of the product of two unnormalised, bell-shaped Student's t-distributions with parameter $\nu = 1$ (also known as Cauchy distribution), which measure how close q^{out} is to n^{out} for input t_1^{in} (sketched in Figure 4.7) and how close to zero for input t_0^{in}

$$p_{\text{th}}(\Phi) = \frac{1}{1 + \left(o + \tilde{n}^{\text{out}} - q^{\text{out}}(\Phi, t_1^{\text{in}})\right)^2} \cdot \frac{1}{1 + \left(q^{\text{out}}(\Phi, t_0^{\text{in}})\right)^2} . \quad (4.34)$$

As with the previous evaluation measure, the semantic value of the pseudo-probabilities is fairly limited in a mathematical sense and should be seen as a normalisation only.

Analogue to similar considerations in the SGSO measure, the use of a heavy-tailed bell-shaped function ensures non-zero $p_{\text{th}}(\Phi)$ with a distinct gradient over vast regions of the parameter space. This gradient might be sufficient to guide an automated parameter optimiser through the neuron parameter space.

The final ingredient to the measure is an estimation of how well the reset condition is fulfilled. Here, the same approach as in the SGSO spike measure in Equation (4.23) is employed

$$p_{\text{reset}}(\Phi) = \exp(-\|\vec{v}_T^1 - \vec{v}_0\|) , \quad (4.35)$$

where $\vec{v}_T^1$ is the state of the neuron at time T for input t_1^{in} and $\vec{v}_0$ is the initial neuron state. Again, the norm $\|\cdot\|$ should rescale the individual state vector components to a comparable value range. The resulting evaluation measure value $\mathcal{P}_{\text{sgmo}}(\Phi)$ is given as

$$\mathcal{P}_{\text{sgmo}}(\Phi) = p_{\text{th}}(\Phi) \cdot p_{\text{reset}}(\Phi) . \quad (4.36)$$

4.5.2 *Fractional spike count*

To tighten the notation, the spike train descriptor t^{in} is not explicitly denoted. Of course, a spike train descriptor must nonetheless be passed to the underlying single neuron simulation.

With the basics of the single group, multiple output spike measure in place, we are ready to address the fractional spike count measure $q^{\text{out}}(\Phi)$ itself. This section first presents a widely unsuccessful approach to the calculation of the fractional spike count. The lessons learned are then incorporated into the construction of a more robust measure. Yet, before we begin, a more thorough definition of the semantics of the seemingly paradoxical "fractional spike count" is appropriate.

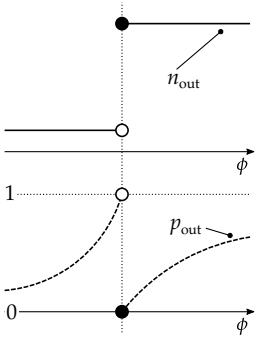


Figure 4.8: Sketch of the fractional spike count decomposition. The fractional spike count measure smoothly interpolates between the discrete steps in n^{out} (top graph) by adding a fractional value p^{out} (bottom graph), which is zero right at the point at which an additional output spike is generated and close to one immediately previous to that point.

Properties of the fractional spike count For the sake of simplicity, let us consider a single variable parameter dimension with value ϕ. All other parameters in the parameter vector Φ should assumed to be constant. Of course, the same concepts apply to more than one variable parameter dimension, yet this would be harder to visualise and requires more elaborate mathematical notation. Furthermore, the postulated properties may be slightly violated by the actually selected fractional spike count measure. They are mere guidelines towards designing such a measure.

The defining property of $q^{\text{out}}(\phi)$ is its decomposability into the discrete spike count $n^{\text{out}}(\phi) \in \mathbb{N}$ and an additive fractional part $p^{\text{out}}(\phi) \in [0, 1) \subset \mathbb{R}$. The latter can be interpreted as the likelihood of an additional spike, which smoothly interpolates between the steps of its discrete counterpart

$$q^{\text{out}}(\phi) = n^{\text{out}}(\phi) + p^{\text{out}}(\phi)\,. \tag{4.37}$$

Given an infinitesimally small $\epsilon \to 0$, the fractional $p^{\text{out}}(\phi)$ should fulfil the following conditions

$$p^{\text{out}}(\phi) = 0 \Leftrightarrow n^{\text{out}}(\phi) > n^{\text{out}}(\phi - \epsilon) \vee n^{\text{out}}(\phi) > n^{\text{out}}(\phi + \epsilon), \tag{4.38}$$
$$p^{\text{out}}(\phi) \to 1 \Leftrightarrow n^{\text{out}}(\phi) < n^{\text{out}}(\phi - \epsilon) \vee n^{\text{out}}(\phi) < n^{\text{out}}(\phi + \epsilon), \tag{4.39}$$

or in other words, the upper corner points of the staircase n^{out} are located on the curve q^{out} (Figure 4.8). In order to facilitate automatic parameter optimisation, p^{out} should be continuous and monotonous for maximally large intervals $[\check{\phi}, \hat{\phi}]$ with

$$n^{\text{out}}(\phi) = n^{\text{out}}(\phi') \quad \text{where} \quad \check{\phi} \leq \phi, \phi' \leq \hat{\phi}\,. \tag{4.40}$$

With these properties in place, we discuss a simple, yet flawed method for the calculation of p^{out}.

4.5.3 *Minimal apical voltage difference*

The method we discuss here is based on an intriguingly simple observation, which in practice fails spectacularly. Consider two infinitesimally

close parameter values ϕ and ϕ', with the value ϕ causing the neuron to produce an additional spike compared to ϕ'. Put more precisely, it holds

$$\tilde{n}^{\text{out}}(\phi) - \tilde{n}^{\text{out}}(\phi') = 1 \quad \text{with } |\phi - \phi'| \to 0\,. \tag{4.41}$$

Repeatedly comparing the membrane potential traces $u(t)$ for such parameter pairs ϕ and ϕ' suggests that the additional spike is very likely to be produced at a time t which corresponds to a local maximum in the membrane potential trace for the parameter ϕ'. Thus, it seems reasonable to assume that a necessary precondition for time points t at which spikes may arise is $\dot{u}(t) = 0$ and $\ddot{u}(t) < 0$. With this thought in mind, the fractional spike count component p^{out} could be defined as follows

The author never encountered a situation in which the spike was not produced at the location of a local maximum. Yet, since the intricate dynamical systems are rather complex, no effort was made at proving this conjecture, so it should be taken with a grain of salt.

$$p^{\text{out}} = 1 - \frac{\min\{E_{\text{Th}}^{\text{eff}} - u(t) \mid \dot{u}(t) = 0 \wedge \ddot{u}(t) < 0\}}{E_{\text{Th}}^{\text{eff}} - E_{\text{L}}}\,, \tag{4.42}$$

which expresses the normalised minimal voltage difference between the local maxima of the membrane potential and the effective threshold potential $E_{\text{Th}}^{\text{eff}}$ (Section 4.4.4). If the maximum membrane potential is close to $E_{\text{Th}}^{\text{eff}}$, it is likely that an additional spike will be introduced, and it holds $p^{\text{out}} = 1$. If the maximum membrane potential is E_{L}, the likelihood of an additional spike is small, so $p^{\text{out}} = 0$.

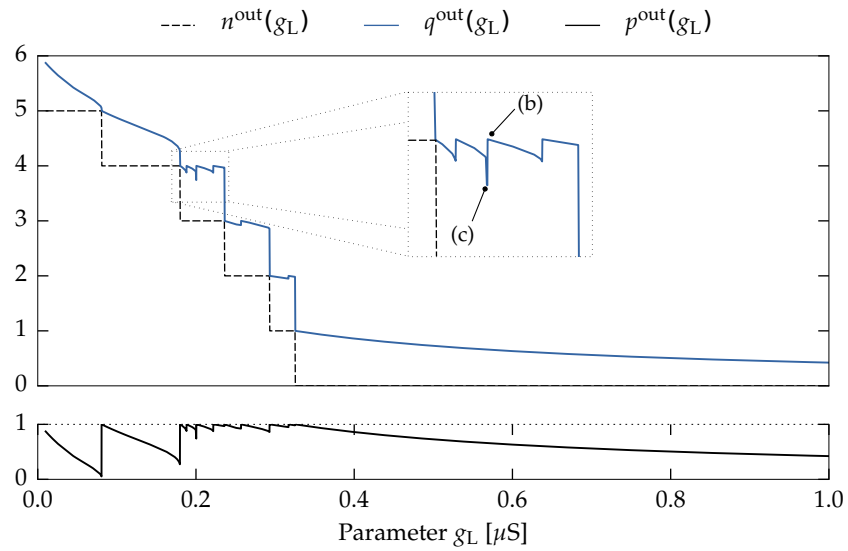
Note that due to discontinuity, the spikes themselves do not fulfil the condition $\dot{u}(t) = 0$. They are thus filtered out.

However, as shown in Figure 4.9, this idea fails on multiple levels. While a maximum in the membrane potential trace at time t for a parameter ϕ' might be a necessary precondition for the production of a spike for ϕ at time t, the reverse does not hold. Even the largest local maximum in the membrane potential trace is by no means a sufficient condition for the production of an *additional* spike for any ϕ. The most spectacular failure of this assumption arises if variation of ϕ indeed produces a spike at the predicted location, but in response a spike at another location disappears.

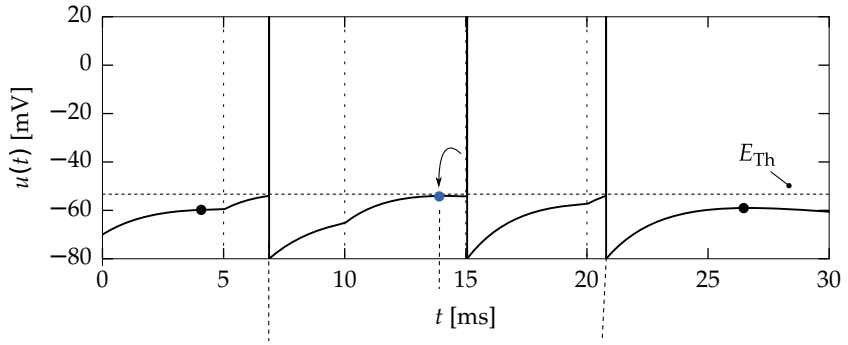
The normalisation of p^{out} poses another problem. For the parameter value ϕ, at which a new spike has just been introduced, Equation (4.38) requires p^{out} to be set to zero. In practice the largest local maximum is larger than E_{L}, causing p^{out} to be set to a value greater than zero.

Nevertheless, the measure possesses at least one positive trait. It produces a smooth curve with subtle slope for $n^{\text{out}}(\phi) = 0$. This is somewhat expected, since most disturbances in the neuron dynamics are generated by the output spike generation mechanism.

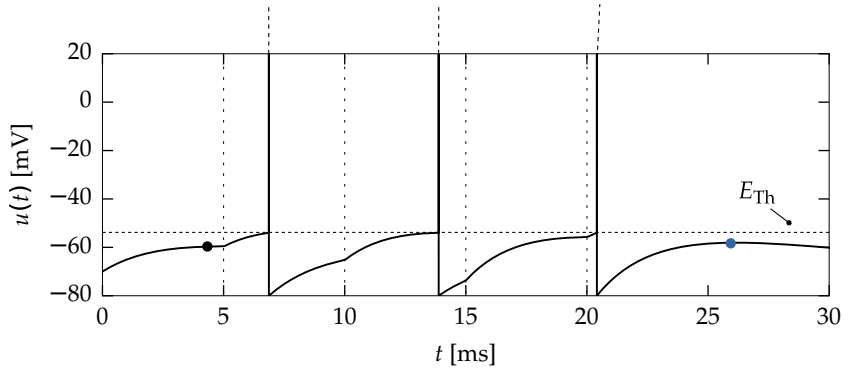
To summarise, it should be apparent from these examples, that running a single neuron simulation and trying to mingle some more or less arbitrarily measured membrane potentials into a single value p^{out} is doomed to fail. Explicit membrane potential measurements are neither expressive, nor properly normalisable. Thus, the next approach is based on a fundamentally different idea.



(a) Minimal apical voltage difference fractional spike count



(b) Neuron membrane potential trace for $g_L = 0.2006\,\mu$S



(c) Neuron membrane potential trace for $g_L = 0.2002\,\mu$S

Figure 4.9: Results for the unsuccessful "minimal apical voltage difference" fractional spike count. (a) plots the number of output spikes $n^{\text{out}}(g_L)$, the fractional spike count measure $p^{\text{out}}(g_L)$ and the pure fractional part $q^{\text{out}}(g_L)$, over the neuron parameter g_L. The fractional spike count is calculated according to Equation (4.42). The tested neuron is a LIF neuron which receives five input spikes in a five millisecond interval starting at $t = 0$. As can be seen in (a), p^{out} exhibits severe discontinuities and non-monotonous behaviours. The figures (b) and (c) show the membrane potential traces $u(t)$ for the values of g_L indicated in (a). Filled circles show the location of the local maxima. As can be seen, the discontinuity in p^{out} is caused by the second spike "jumping" to the position of the largest maximum (blue circles).

4.5.4 *Minimal membrane potential perturbation*

The notion of "likelihood of an additional spike" could be reformulated as "what is the minimal amount of additional energy that must be pumped into the neuron to produce another spike?". An obvious way of injecting energy into a spiking neuron is to offset the membrane potential at a certain time t by a small voltage Δu. The basic idea of the approach to the fractional spike count measure presented here is to find the minimal Δu, which introduces an additional output spike. However, two open questions remain: at which times t should the perturbation take place and what is the possible value range of Δu?

Searching a minimal Δu over a dense grid of times t is not a viable option, as this approach is not only slow, but would inflate the search space unnecessarily since the neuronal behaviour usually does not differ tremendously between two close points in time t and t'. The events which influence the neuronal dynamical system the most are the output spikes themselves. The membrane potential perturbation is thus performed at the times of highest behavioural uncertainty, namely, at the beginning of the simulation, and the end of each refractory period in the original, undisturbed neuron simulation.

The possible value range for perturbations at time t is limited by the original membrane potential $u(t)$ and the effective threshold potential $E_{\mathrm{Th}}^{\mathrm{eff}}$. It must hold $\Delta u(t) \in [0, E_{\mathrm{Th}}^{\mathrm{eff}} - u(t)]$. Given these considerations, the entire fractional spike count calculation can be expressed mathematically as

$$p^{\mathrm{out}}(\Phi) = 1 - \min \left\{ \frac{\Delta u}{E_{\mathrm{Th}}^{\mathrm{eff}} - u(t + \tau_{\mathrm{ref}})} \middle| n_{\mathrm{pert}}^{\mathrm{out}}(\Phi, t + \tau_{\mathrm{ref}}, \Delta u) > n^{\mathrm{out}}(\Phi), \right.$$
$$\left. \Delta u \in [0, E_{\mathrm{Th}}^{\mathrm{eff}} - u(t + \tau_{\mathrm{ref}})], t \in (-\tau_{\mathrm{ref}}) \| t^{\mathrm{out}} \right\}, \qquad (4.43)$$

where $n_{\mathrm{pert}}^{\mathrm{out}}(\Phi, t, \Delta u)$ is the output spike count for perturbation with offset Δu at time t, t^{out} are the original output spike times, and $u(t)$ refers to the original membrane potential trace.

The gist of the algorithm as a whole is captured in Equation (4.43). Exact descriptions of the optimised fractional spike count algorithm are out of scope for this thesis. Interested readers can find the entire implementation in the accompanying source code. For example, an especially crude trick was used for efficiently relaying Δu to the neuron simulator: t and Δu are encoded as a special input spike with Δu stored in the payload of a "NaN" weight value.

Algorithmic implementations of Equation (4.43) should take various optimisations into account. Most importantly, the search for the minimum Δu at time t should be implemented as a binary search. Furthermore, since neuron dynamics only need to be simulated starting from the time-point of the perturbation t, the algorithm should start its minimum search at the largest t, and thus successively restrict the search space for longer simulation durations. The next optimisation resembles the concept of dynamic programming [Bel57]. For each tested time t, the neuronal states $\vec{v}$ which did or did not lead to an additional output spike are stored in a table. Whenever the simulator passes t it compares the current neuron state to the corresponding table entry allowing for early abortion. This approach has to be implemented with great care, since all neuron states – including for example the

adaptation current $I_a(t)$ – influence the generation of additional output spikes.

As shown in Figure 4.10, the perturbation-based method is clearly superior to the previous concept – at least for regions with $n^{\text{out}} > 0$. For $n^{\text{out}} = 0$ p^{out} (black) falls very quickly to or stays at zero, as can be seen for example in the bottom portion of Figure 4.10(d). The pragmatic solution taken here is to employ the previous measure (denoted as $p^{\text{out}\prime}$ in the figure) in regions with $n^{\text{out}} = 0$. In combination (blue graphs, q^{out}), the two measures fulfil the behavioural constraints postulated above to a high degree. The pure fractional part of the measure p^{out}, which is mostly monotonous between steps, fills the entire range between zero and one, and touches the upper corners of the underlying step function n^{out}. Minor deficiencies concern the non-linear interpolation between the steps, the generation of plateaus as in Figure 4.10(f), and very small, non-monotonous kinks as in Figure 4.10(b). However, since an optimiser is most likely to adapt multiple dimensions concurrently, at least one gradient should point into the correct direction. Neither the plateaus, nor the occasional kink should thus present a serious problem.

4.6 NEURON EVALUATION SOFTWARE FRAMEWORK

This section gives insight into the software framework and tools for single neuron evaluation experiments. Following an overview of the system architecture, the interactive parameter space exploration tool *AdExpSimGui* and in particular the modular high performance neuron simulator are discussed.

4.6.1 *Architectural overview*

The entire framework, including the CLI and GUI applications, amounts to more than 18 000 *lines of code, including inline documentation. The core library is solely based on the C++ standard library and can be used on any system with a standard compliant C++14 compiler.*

The single neuron simulator and the three evaluation methods are implemented as part of the *AdExpSim* framework in the C++14 programming language [ISO14]. Architecture and components of the framework are sketched in Figure 4.11. Low-level facilities, such as the generation of spike trains, normalisation of neuron parameters and the LIF and AdEx neuron model simulator, are implemented as part of a core library. On a higher abstraction level, the same library implements the neuron evaluation measures presented in the previous sections, as well as classes for multi threaded parameter sweeps (exploration) and optimisation. The input and output (I/O) library provides functions for serialisation of the user-provided parameters to JSON (JavaScript Object Notation, [Bra14]), as well as facilities for the export of exploration data as surface plots.

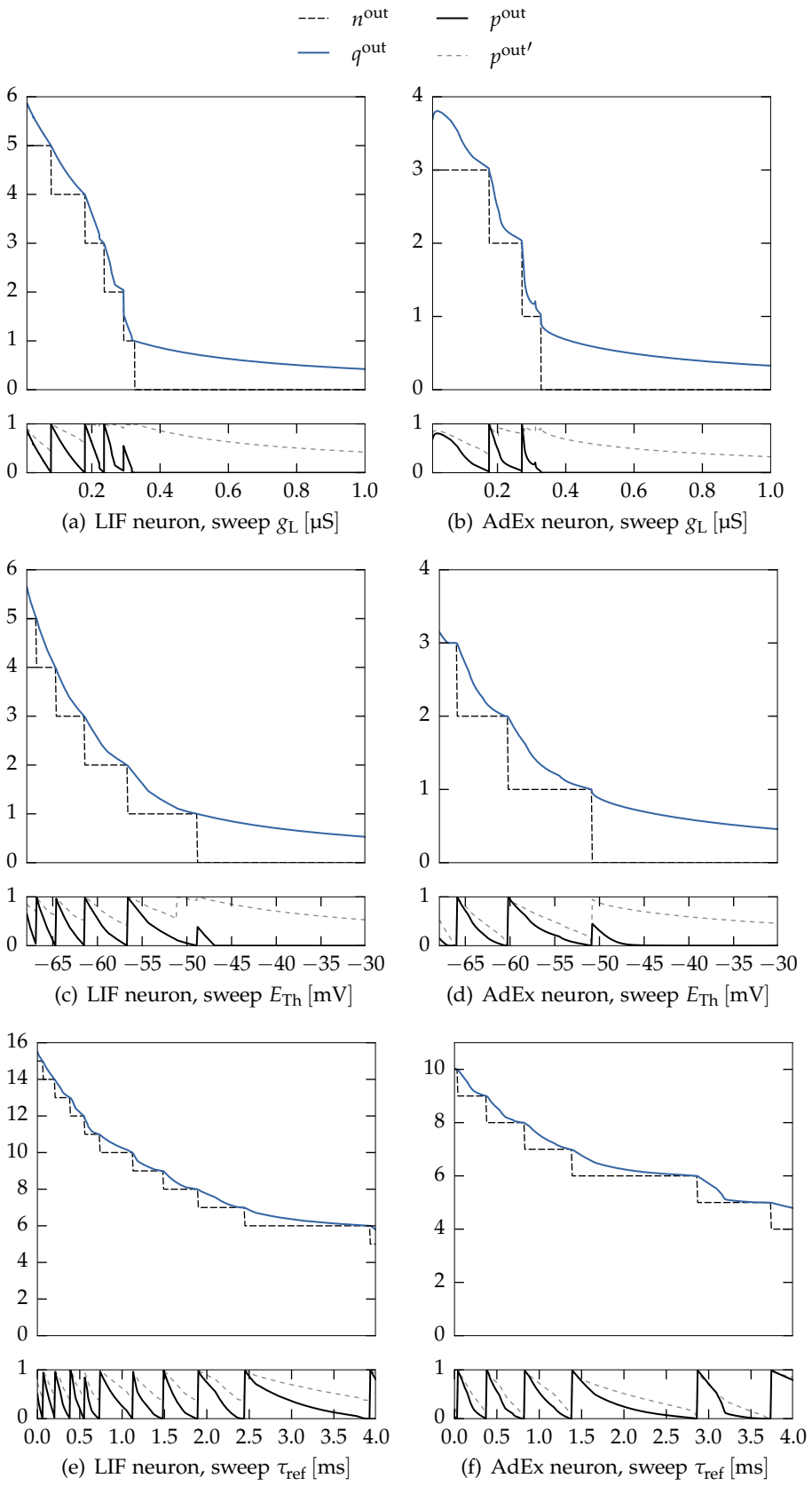


Figure 4.10: Minimal membrane potential perturbation measure examples, comparison between LIF (left) and AdEx (right) neurons with sweeps over different parameter dimensions.

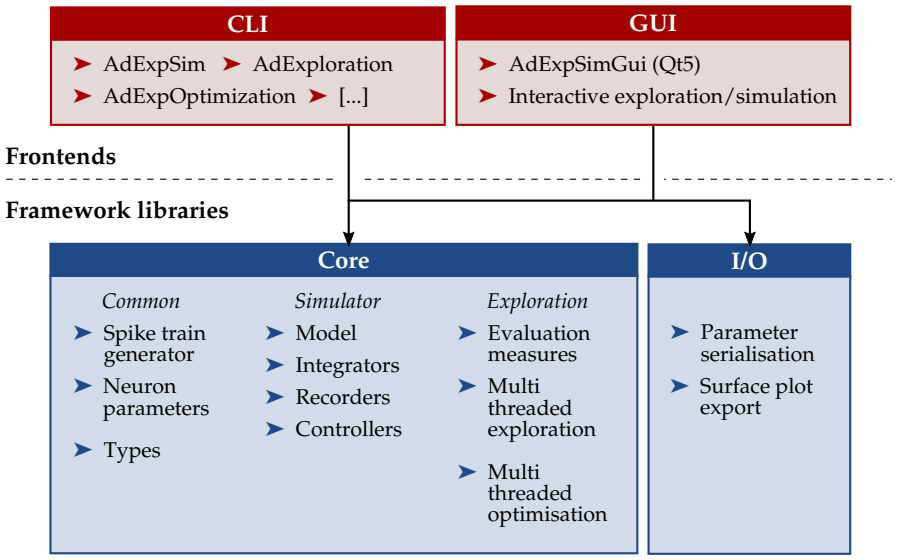
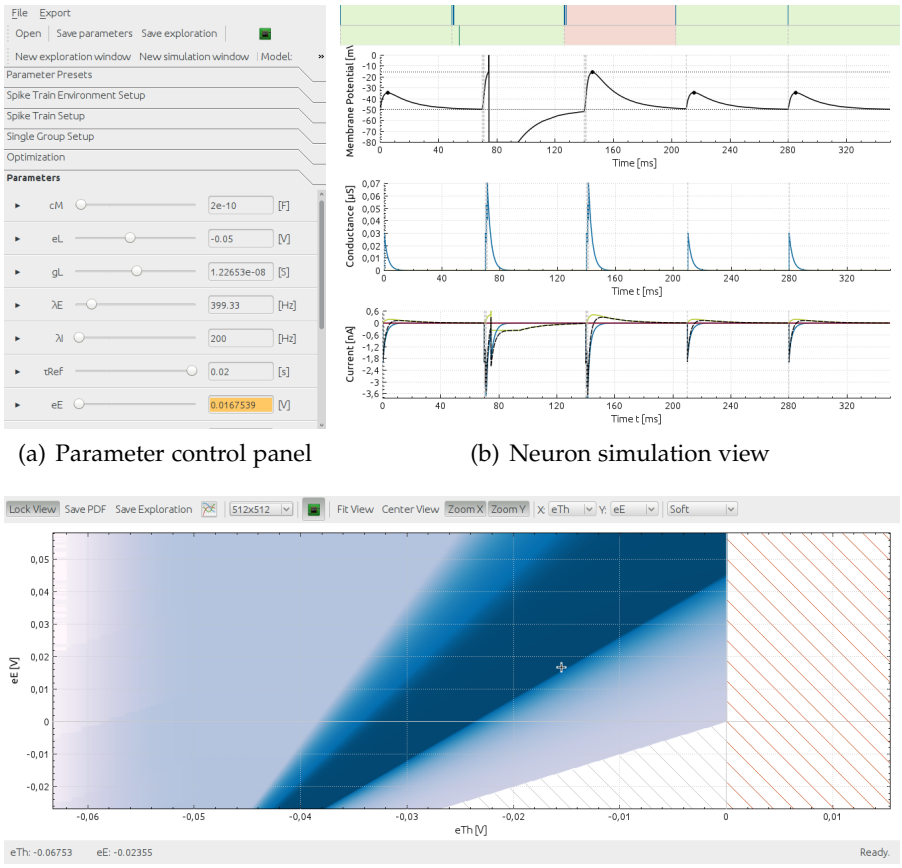


Figure 4.11: Architectural overview of the *AdExpSim* framework developed for this thesis with distinction between the frontend applications and the actual framework libraries, which implement the listed functionalities in a reusable manner.

4.6.2 *Frontend applications*

The *AdExpSim* framework comes with a set of frontend applications. The command line interface (CLI) applications are too numerous to list in their entirety, but are usually thin wrappers around the core library which expose an individual aspect of the core functionality to end users. These programs require no interaction and facilitate batch processing of exploration and optimisation tasks. Most of the programs are designed to run a specific experiment and can be configured directly in the source code.

The *AdExpSimGui* graphical user interface (GUI) application is an interactive tool for design space exploration. The application itself is based on the *Qt5* application framework and the *QCustomPlot* plotting library [Qt 15; Eic15]. The control panel shown in Figure 4.12(a) allows users to setup neuron and evaluation measure parameters. Results of a spike train evaluation for the current parameter vector, along with the corresponding neuron voltage, conductance and current traces are displayed in the simulation window depicted in Figure 4.12(b). Two dimensional projections of the high-dimensional neuron parameter space can be viewed in one or multiple exploration windows (Figure 4.12(c)). The user can interactively pan and zoom the design space, and select the evaluation measure and the design space dimensions. Changes to the neuron parameters are reflected in all exploration windows at the same time. The exploration view highlights invalid parameter combinations and parameters outside the range supported by NM-PM1.



(a) Parameter control panel (b) Neuron simulation view

(c) Design space exploration view

Figure 4.12: Screenshots of the *AdExpSimGui* tool. (a) depicts the parameter control panel, (b) shows the spike train neuron simulation window, and (c) shows a design space exploration window. Dark colours correspond to higher-rated parameters vectors, the red hatching to parameters outside the hardware range, the grey hatching to invalid parameter combinations.

4.6.3 *High performance single neuron simulator*

The demands regarding the functionality of the high performance single neuron simulator at the core of the *AdExpSim* framework vary greatly between the individual evaluation measures. An incomplete list of features includes the recording of voltage, current and conductance traces, the tracking of local maxima, early abortion, simulation of both the AdEx and LIF models, usage of various integrators, deactivation of the spiking mechanism, as well as injection of perturbation voltages at defined times. To this end, the simulator is implemented as a C++ template. This allows building individual simulator instances tailored specifically to a feature set. The compiler can optimise each instance to a high-performance and cache-oblivious loop consisting of a few hundred assembler instructions. See Appendix A.1 for technical details.

EXPECTED EVALUATION MEASURE BEHAVIOUR				
	SPIKE TRAIN		SGSO	SGMO
	Small n_g	Large n_g		
Reproducibility	$-^*$	$\circ^*$	$+$	$+$
Efficiency	$\circ$	$-$	$+$	$\circ$
Interpretability	$+$	$+$	$-$	$-$
Realism	$+$	$+$	$-$	$\circ$
Smoothness	$-$	$\circ$	$+$	$+$
Allows $\tilde{n}^{\text{out}} > 1$	✓	✓		✓
Stochastic input	✓ *	✓ *		

Table 4.7: Informal classification of the expected behaviour for the three evaluation methods presented in this chapter ($+$ *good*, $\circ$ *mediocre*, $-$ *bad*). Refer to the text for more information. $*$ Of course deterministic input spike trains can be used, however, this reduces the realism of the measure.

4.7 EVALUATION METHOD COMPARISON

This section aims at providing a direct comparison of the three evaluation measures. First, the expected properties of the measures are summarised. Then, experiments and results concerning their actual output and their suitability for optimisation are presented.

4.7.1 *Evaluation measure properties*

Important properties of the evaluation measures are their determinism or *reproducibility*, the computational *efficiency*, the *interpretability* of the results with respect to the full network evaluation measures, the *realism* of the measure regarding the actual conditions in a spiking BiNAM network, and the *smoothness* of the function for variations in the parameter space Φ. Table 4.7 informally summarises the properties of the individual measures. These experimental hypotheses are elaborated in the following.

For a small number of experiment groups $n_g \ll 100$ and assuming non-zero noise parameters, the spike train measure is non-deterministic, yet expected to be reasonably fast. The measure possesses a clear interpretation, namely the probability with which the threshold and reset condition are fulfilled. As it samples directly from the input model and simulates all aspects of the neuron, it can be regarded as realistic. The output of the measure contains discrete steps. For $n_g \gg 100$ the reproducibility increases, while the size of the individual steps in the output decreases at the cost of a higher simulation time.

Both the SGSO and SGMO measure are reproducible and provide smooth output. The pseudo-probabilistic values of the $\mathcal{P}_{\text{sgso}}$ and $\mathcal{P}_{\text{sgmo}}$

INITIAL NEURON PARAMETERS

Synapse				*Membrane (LIF)*				*AdEx*			
w	=	0.03	µS	C_m	=	1.00	nF	a	=	4.00	nS
E_e	=	0.00	mV	g_L	=	0.08	µS	b	=	80.50	pA
τ_e	=	5.00	ms	E_L	=	−70.00	mV	τ_a	=	144.00	ms
				E_Th	=	−54.00	mV	E_Th	=	20.00	mV
				E_reset	=	−80.00	mV	$E_\mathrm{Th}^\mathrm{exp}$	=	−54.00	mV
				τ_ref	=	0.00	ms	Δ_Th	=	2.00	mV

Table 4.8: Initial neuron parameters as categorised in Table 4.1, for the synapse, and the LIF and AdEx neuron. The specified LIF parameters are also used for the AdEx neuron, with exception of E_Th.

compound measures bears hardly any information that can be directly translated to the performance of the neuron in the BiNAM, it solely describes an abstract "optimality". The SGSO measure discards potentially important parts of the neuron model and its environment, and is therefore expected to be the least realistic. It requires idealised deterministic input spike trains, a deactivated spiking mechanism, idealised assumptions regarding the effective threshold potential, and a capped threshold current. The SGMO measure simulates all aspects of the neuron, yet its realism is still limited by the use of the deterministic input spike train.

4.7.2 *Empirical comparison*

The above characterisations are hypotheses for the expected behaviour of the single neuron evaluation measures. In this section we conduct and discuss parameter sweeps in order to empirically compare the properties of the single neuron evaluation measures.

Method Two two-dimensional parameter sweeps (explorations) are performed, each for a LIF and an AdEx neuron. The first sweep varies g_L from 0.01 µS to 0.6 µS and τ_e from 1 ms to 100 ms. The second sweep varies E_Th from −67.9 mV to 0 mV and w from 0 µS to 1 µS. The grid resolution of the exploration is set to 1024, resulting in a total of 1 048 576 parameter evaluations. The neuron must output a single spike for $n_1 = 3$ input spikes and no output spike for $n_0 = 2$ input spikes. See Table 4.8 and *Scenario I* in Table 4.10 for the complete set of neuron and system parameters. The spike train measure is executed with two experiment group sizes, $n_\mathrm{g} = 10$ and $n_\mathrm{g} = 100$, referred to as ST_{10} and ST_{100} in the following.

$g_\mathrm{L} = 0\,\mu\mathrm{S}$ and $\tau_\mathrm{e} = 0$ are not valid according to the design space constraints (Section 4.1.4). Slightly larger values were chosen as a starting point instead. Analogously, the initial value for E_Th is chosen as $E_\mathrm{L} + \Delta_\mathrm{Th} + 1\,\mathrm{mV}$ to fulfil the constraints.

Results Table 4.9 shows the runtimes of the individual parameter sweeps. Independent of the measure, there is an approximate factor

NEURON PARAMETER SPACE EXPLORATION RUNTIMES				
	Sweep over g_L and τ_e		*Sweep over E_{Th} and w*	
	LIF	ADEX	LIF	ADEX
ST_{10}	648 s	1595 s	4368 s	12 595 s
ST_{100}	6578 s	13 890 s	45 840 s	125 000 s
SGSO	206 s	310 s	2942 s	4844 s
SGMO	358 s	1014 s	7721 s	15 720 s

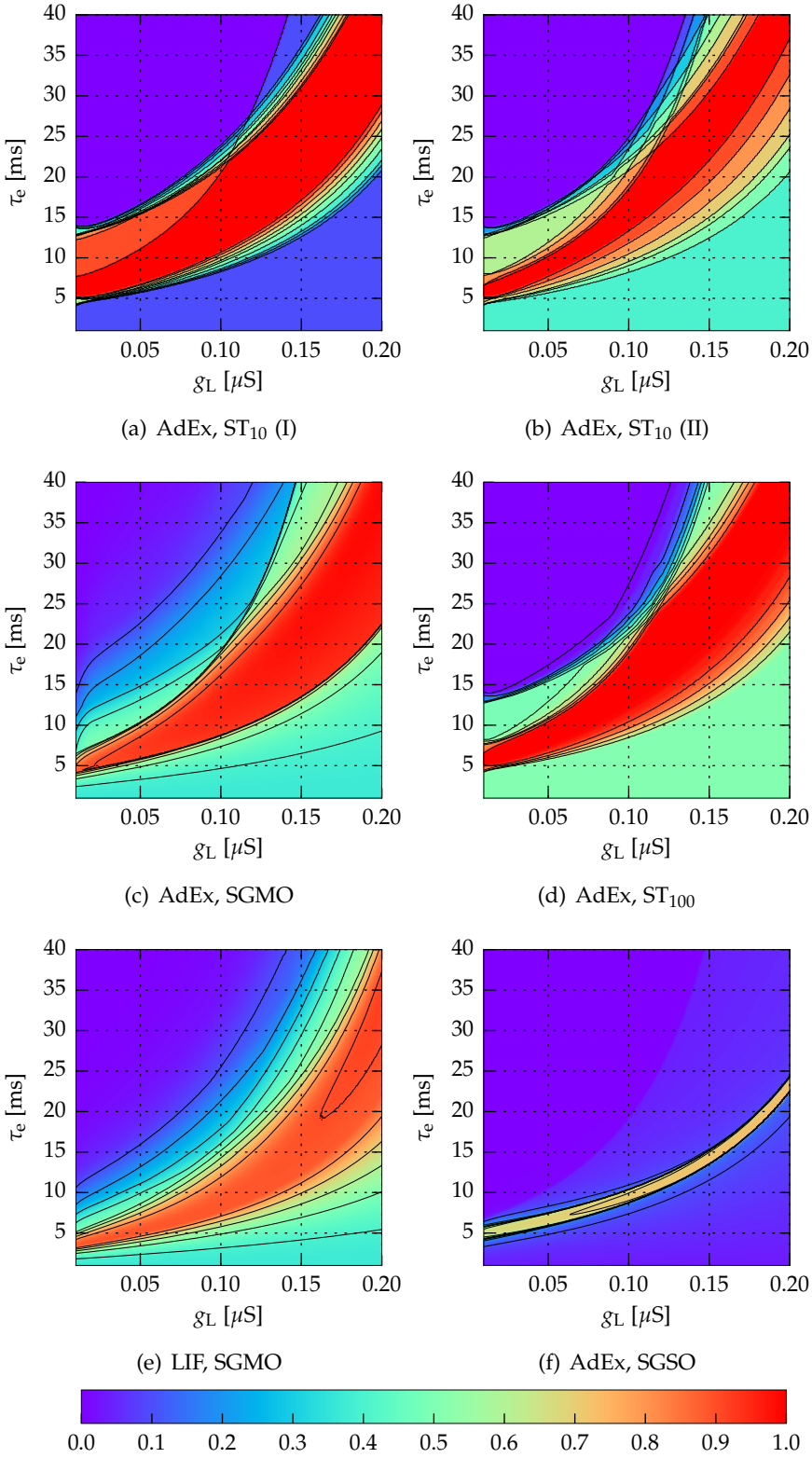
Table 4.9: Neuron parameter space exploration experiment runtimes for a 1024×1024 parameter sweep. The shown times correspond to the total CPU time, the wall-clock time is about $1/24$ of the given values. The results for the spike train measure with $n_g = 10$ are averaged over two runs.

two in runtime between the LIF and the AdEx neuron evaluation. The second sweep (over E_{Th} and w) is about ten times slower than the first. SGSO is clearly the fastest measure, being two-to-three times faster than SGMO, which in return is two times faster than ST_{10} in the first exploration run, and vice-versa in the second exploration. Unsurprisingly, ST_{100} is ten times slower than ST_{10}.

An interesting effect that can be found in Appendix C is noise produced by the ST_{100} measure in conjunction with the AdEx neuron for large g_L and small τ_e. This is believed to be caused by the adaptive step size integrator in conjunction with the fast neuron dynamics in this region. The author assures that this kind of noise is not unique to the spike train measure. If desired, it can be eliminated by employing a constant step size integrator at the cost of magnitudes larger runtimes.

Figure 4.13 shows visualisations of the evaluation measures for subregions of the first parameter sweep. Figures 4.13(a) and 4.13(b) show two independent runs of ST_{10} for an AdEx neuron. The results differ significantly from each other. There are clearly visible discrete steps in the result. Upon first visual inspection, the SGMO measure in Figure 4.13(c) replicates ST_{100} in Figure 4.13(a), while featuring a clear gradient leading towards the centre region with high optimality. In contrast, ST_{100} is uniform in vast regions of the design space. The gradient of the SGMO measure is even clearer in Figure 4.13(e), where the result for the LIF neuron is shown. Interestingly, the region with maximum optimality is more concentrated than in the results for the AdEx neuron. The result for the SGSO measure in Figure 4.13(f) is visually distinct from the others, yet the location of the region with high optimality is largely the same. The SGSO measure furthermore shows a subtle gradient. Upon closer inspection it becomes apparent that the maximum regions of the SGMO and SGSO measures do not entirely cohere with those of ST_{100}. They seem to extend a little further towards larger g_L. The complete set of results including the second exploration can be found in Appendix C. Similar to the manually picked examples shown here, all measures exhibit a pronounced region with maximum optimality $\mathcal{P}$. Unsurprisingly, SGSO and SGMO feature a clear gradient, whereas ST_{100} does not.

Discussion A plausible explanation of the runtime differences between the individual exploration runs and the AdEx and LIF neuron is the adaptive step size integrator. The exponential spike generation mechanism of the AdEx neuron apparently requires smaller integra-



(a) AdEx, ST_{10} (I)

(b) AdEx, ST_{10} (II)

(c) AdEx, SGMO

(d) AdEx, ST_{100}

(e) LIF, SGMO

(f) AdEx, SGSO

Figure 4.13: Selected results of the first design space exploration sweep (over g_L and τ_e). Colour-coded is the evaluation result $\mathcal{P} \in [0, 1]$. The contour lines correspond to the ticks on the colour bar. Only a portion of the design space exploration is shown. Refer to the text for more details.

tion steps, causing a higher runtime. The sweep over E_{Th} is likely to produce many output spikes for small E_{Th}, which explains the additional slowdown and especially the disproportionate runtime increase for the SGMO measure: here, the runtime directly depends on the number of output spikes. The differences in runtime should be seen as a strong argument for the adaptive step size integrator, as it is responsible for the major speedups in parameter space regions with few output spikes.

The empirical results of the evaluation measures themselves fit well into the conjectures made in Section 4.7.1. Most interestingly, the SGMO measure indeed provides a very good approximation of the ST_{100} measure at a fraction of a runtime, as well as a smooth gradient. Compared to the AdEx neuron, the measures for the LIF neuron model exhibit more smooth results, which is likely due to the exponential attractor dynamics in the AdEx neuron. The visually distinct appearance of the SGSO measure is likely caused by the multiplication of four terms in Equations (4.22) to (4.24), which, despite the heavy-tailed sigmoids, tends to generate small values. Careful tuning of the underlying distribution meta-parameters may improve the predictive power of both the SGSO and the SGMO measure.

4.7.3 *Automated parameter optimisation*

The SGMO and SGSO measures exhibit a smooth gradient in two dimensional parameters sweeps, like those shown above. The gradient could be followed by a simple optimisation algorithm to find the global subspace maximum. It is doubtful, that this property translates to higher-dimensional subspaces, where it is likely that local maxima are omnipresent. Furthermore, the claim, that the gradient supports parameter optimisation has not been tested yet. In the following, we perform a benchmarking experiment which tests the suitability of the single neuron evaluation measures for the task they were designed for – parameter optimisation.

Automated parameter optimisation itself is not the focus of this thesis and surly more advanced experiments and optimisation methods should be employed in future work.

Methods Goal of the experiment is to optimise the parameters of an LIF and an AdEx neuron with respect to the three single neuron evaluation measures, ST_{100}, SGSO and SGMO. The system parameters are chosen according to three scenarios presented in Table 4.10. The first scenario tests single output spikes, the second bursts of $s^{\mathrm{in}} = s^{\mathrm{out}} = 3$ spikes, and the third a neuron population of $K = 3$ neurons. For the LIF neuron, the only potential being optimised is E_{Th}, for the AdEx neuron $E_{\mathrm{Th}}^{\mathrm{eff}}$. Furthermore, C_{m} is kept constant. Starting point of the optimisation are the initial neuron parameters Φ shown in Table 4.8.

The optimisation method itself is a standard Nelder and Mead minimisation algorithm, a multidimensional generalisation of the bisection method also known as *Downhill Simplex* [NM65]. Advantages of the

SCENARIOS FOR NEURON PARAMETER OPTIMISATION				
Relevant network parameters				
		SCENARIO I	SCENARIO II	SCENARIO III
Ones in the input	c	3	3	3
Population size	K	1	1	3
Data encoding parameters				
		SCENARIO I	SCENARIO II	SCENARIO III
Input burst size	s^{in}	1	3	3
Output burst size	s^{out}	1	3	3
Interspike interval	Δt	10 ms	10 ms	10 ms
Sample interval	T	200 ms	200 ms	200 ms
Noise parameters				
		SCENARIO I	SCENARIO II	SCENARIO III
Jitter	σ_t	5 ms	5 ms	5 ms
Resulting input and output spike counts				
		SCENARIO I	SCENARIO II	SCENARIO III
Upper threshold	n_1	3	9	27
Lower threshold	n_0	2	6	18
Output spike count	$\tilde{n}^{\text{out}}$	1	3	3

Table 4.10: Scenarios for the neuron parameter optimisation test. The table shows the chosen system parameters (Table 4.1) and the resulting input and output target spike counts. The noise parameters p_1, p_0, σ_t^{offs}, σ_w are set to zero since they cannot be tested with the SGSO and SGMO measure, or, in the case of σ_t^{offs}, are not explicitly modelled. System parameters not listed here are irrelevant for single neuron evaluation.

algorithm are its simplicity and implicit gradient calculation. A major disadvantage is its slow convergence [Pre+07b]. The implementation provided as part of *AdExpSim* features concurrent parallel execution of slightly varying vectors and automated restart for the escape from local minima, support for arbitrary parameter constraints and discrete parameters, such as w in NM-PM1 and Spikey. Apart from this, the algorithm follows the implementation presented in Numerical Recipes [Pre+07b].

Results A striking property of the results shown in Table 4.11 is incoherence of the evaluation measure optimality values. Optimising with ST_{100} as target measure results in optimality values below those for unoptimised parameters for the SGMO and SGSO measures. Optimisation with SGSO results in zero optimality according to the SGMO measure for the LIF neuron. Yet, parameter vectors optimised with SGSO receive a high lateral score from the ST_{100} measure. Optimisation for SGMO results in mediocre values for both ST_{100} and SGSO. Still at least for *Scenario I*, optimisation significantly increases the optimality value of the current target measure (bold values in the

INITIAL EVALUATION MEASURE OPTIMALITY VALUES							
Scenario		*LIF*			*AdEx*		
		ST_{100}	SGSO	SGMO	ST_{100}	SGSO	SGMO
Initial	I	0.78	0.43	0.89	0.46	0.11	0.51
	II	0.28	/	0.22	0.00	/	0.16
	III	0.00	/	0.00	0.00	/	0.00
FINAL EVALUATION MEASURE OPTIMALITY VALUES							
Target measure/		*LIF*			*AdEx*		
Scenario		ST_{100}	SGSO	SGMO	ST_{100}	SGSO	SGMO
ST_{100}	I	◇ **1.00**	0.40	0.76	**1.00**	0.01	0.09
	II	**0.52**	/	0.00	**0.46**	/	0.00
	III	**0.00**	/	0.00	**0.39**	/	0.00
SGSO	I	1.00	◇ **0.61**	0.00	0.92	**0.70**	0.91
SGMO	I	0.73	0.33	◇ **0.98**	0.56	0.50	**0.92**
	II	0.42	/	**0.94**	0.00	/	**0.19**
	III	0.44	/	**0.97**	0.00	/	**0.00**

Table 4.11: Initial and final neuron evaluation measure optimality values. The top part of the table shows the initial evaluation measure optimality values $\mathcal{P}$ prior to optimisation, with respect to the three system parameter scenarios (rows). The bottom part of the table shows the same evaluation measures after the parameters have been optimised with respect to a scenario and target optimisation measure (rows). Bold values correspond to the final values of the target measure. ◇ The final neuron parameter vectors for these results are shown in Table 4.12.

OPTIMISED LIF NEURON PARAMETERS							
				INIT.	ST_{100}	SGSO	SGMO
	Membrane cap.	C_m	[nF]	1.00	1.00	1.00	1.00
○	Leak conductance	g_L	[nS]	50.00	45.50	13.40	14.50
	Leak potential	E_L	[mV]	-70.00	-70.00	-70.00	-70.00
○	Threshold potential	E_{Th}	[mV]	-54.00	-52.50	-48.23	-65.80
	Reset potential	E_{reset}	[mV]	-80.00	-80.00	-80.00	-80.00
○	Refractory period	τ_{ref}	[ms]	0.00	0.00	0.00	0.00
	Synapse potential	E_e	[mV]	0.00	0.00	0.00	0.00
○	Synapse time	τ_e	[ms]	5.00	5.37	0.10	1.20
○	Synapse weight	w	[nS]	30.00	36.25	1477	19.23

Table 4.12: Initial and optimised LIF neuron parameters. The corresponding optimality values $\mathcal{P}$ are shown in Table 4.11, where they are marked with a "◇"-symbol. ○ These parameter dimensions were optimised.

table). Remarkably, the SGMO measure is capable of finding "optimal" solutions for the second and third scenario in conjunction with a LIF neuron, starting from a small optimality values.

Table 4.12 shows the optimised LIF neuron parameters for the first system parameter scenario. The chosen parameters differ significantly between the target measures. The SGSO measure converges towards a rather peculiar parameter combination, with a very small synapse time constant τ_e, small leak conductance g_L, and a relatively large synapse weight. Yet, despite their differences, this parameter vector and the parameter vector optimised by ST_{100}, are evaluated with maximum optimality by ST_{100}.

Discussion The incoherence of the optimality values is most likely influenced by three factors: deficiencies of the evaluation measures, a lack of regularisation, and different semantics of the values. Problems with the evaluation measures have already been observed in the previous experiments, where SGSO and SGMO do not entirely overlap with ST_{100}. The lack of parameter regularisation allows grotesque parameter combinations, as those found for the LIF neuron and *Scenario I*, which in return are likely to be rejected by other evaluation measures. The semantics of the ST_{100} measure are entirely different from SGSO and SGMO. A non-spiking neuron will always produce a value of $\mathcal{P}_{st} = 0.5$, as half of all spike groups are successful. Such a behaviour would be scored with a near zero value by SGSO and SGMO. On the other hand, the spike train measure is extremely harsh, as any deviation in the output spike count of just a single spike out of multiple is counted as failure.

Despite these issues, the results clearly show that parameter optimisation is indeed feasible with the presented methods. All measures find optimal (with respect to themselves) parameter combinations for *Scenario I*. The fact that optimisation with SGMO finds such parameters for the LIF neuron in the intrinsically difficult scenarios *II* and *III* hints at the power of smooth gradients in the parameter space in conjunction with the chosen optimisation method. Nevertheless, further improvement of SGMO is inevitable, since the parameters will not fulfil the BiNAM requirements in practice ($\mathcal{P}_{st} < 0.5$). As long as the neuron should only produce a single output spike, the SGSO measure seems to be a viable option for parameter selection.

The ST_{100} measure only works as well as presented, because the optimiser probabilistically reinitialises the parameter vectors once they converge, which causes the optimiser to eventually "jump" over the discrete steps in ST_{100}.

4.8 CONCLUSION

In this chapter we described the design space, and presented and compared three approaches to single neuron evaluation. A comprehensive software framework has been developed for the simulation and evaluation of single neurons. Two-dimensional parameter sweeps over parts of the design space show a rather well-behaved landscape with pronounced, coherent regions with high optimality. Nevertheless, the

parameter optimisation experiment demonstrates the perils of high-dimensional parameter spaces. Although the SGMO measure does not perform as well as anticipated, it and its underlying fractional spike count measure seem to be viable approaches to neuron parameter optimisation.

Until now, the predictive power of the single neuron evaluation measures have not been examined regarding the ground truth, namely full network simulation. This examination is one of the goals of the next Chapter 5, in which we finally execute the BiNAM on actual neuromorphic hardware and – amongst others – compare the full network evaluation results with the predictions from the single neuron evaluation.

5 FULL NETWORK SIMULATION EXPERIMENTS

This chapter puts the previous theoretical considerations regarding BiNAM design space exploration and full network evaluation into practice. We finally execute full spiking Willshaw associative memory networks on neuromorphic hardware, assess the predictive power of the single neuron evaluation measures presented in the previous chapter and compare experimental results obtained from different platforms.

In Section 5.1, we discuss the overall methodology for full network simulation. This includes the software framework implemented to this end and an overview of current hardware limitations. Section 5.2 revisits the neuron parameter sweeps presented in Section 4.7.2. In contrast to the previous experiments, the sweeps are now backed by entire memory networks, either executed on neuromorphic hardware or in software simulation. Finally, Section 5.3 presents an analysis of the suitability of BiNAM system parameter variation as a possible neuromorphic system benchmark.

5.1 METHODOLOGY AND SOFTWARE ARCHITECTURE

In theory, the methodology for the simulation and evaluation of full BiNAM networks is rather straightforward. Given system and neuron parameters Φ (Section 4.1.1), a test dataset $\mathfrak{D}$ is generated according to Algorithm 3.3. A BiNAM memory matrix M is then calculated for this dataset (Equation (2.27)). The memory matrix is the basis for the synaptic connections in the network graph (Section 3.1) and the input spikes train descriptors t^{in}, k^{in} (Algorithm 3.1). The network is then executed with the help of the *PyNN* abstraction layer (Section 2.4.4) on an arbitrary hardware or software simulator. Finally, the recorded output spike trains t^{out} are analysed with respect to the evaluation measures presented in Section 3.2.

For two reasons this approach is not feasible in practice: the first being platform-specific inconsistencies in the *PyNN* software layer, and the second an inefficient utilisation of the hardware resources. Two software based solutions were developed to overcome these challenges, *PyNNLess* and *PyNAM*, which are presented in the following.

5.1.1 *PyNNLess*

There are – often minor – inconsistencies in the implementation of the *PyNN* specification for all targeted platforms, for both hardware and software simulators. Furthermore, a total of three consecutive *PyNN* versions have to be supported. Each version differs in functionality and

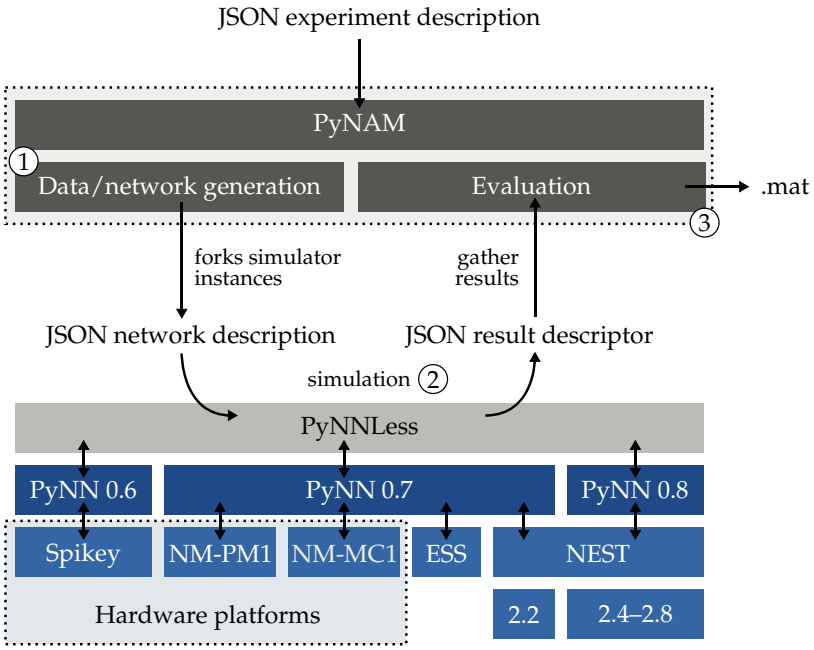


Figure 5.1: Overview of the software architecture for full network evaluation. The *PyNAM* tool generates a series of the network graphs according to a JSON experiment description file. The serialised, JSON-like network descriptors are passed to *PyNNLess*, a platform agnostic wrapper around the *PyNN* framework, which in return passes the recorded results back to *PyNAM*, where they are analysed and stored in an HDF5-file (`.mat`). The encircled numbers refer to the *PyNAM* execution stages.

The encountered inconsistencies are most likely due to the early state of the HBP software stack and are likely to be solved in the next months. The experiments presented in this thesis are the first to run on all HBP hardware platforms.

semantics of the provided API. This renders the use of the exact same code across all platforms difficult. Generally, there are two possible solutions to such a problem. Either the network execution stage in the above pipeline is rewritten for each target platform, or another software abstraction layer is added on top of *PyNN*, which levels the discrepancies out.

A short usage example of PyNNLess *is provided in Appendix A.2.*

To avoid redundancies and improve the modularity of the code, the latter approach was chosen for the implementation of experiments on neuromorphic hardware for this thesis. The result of this effort is the *PyNNLess* Python library. Its functionality is provided as a data-driven API [Red11]. The API receives a network descriptor as a simple data structure, which consists of the neurons and their connectivity as adjacency list, the input spike trains and the signals for recording. After successful network execution, the library returns a data structure containing the recorded output. *PyNNLess* supports all HBP neuromorphic hardware platforms, as well as the NM-PM1 emulator ESS and the software simulator NEST (Figure 5.1).

5.1.2 *PyNAM*

Construction and evaluation of single BiNAM instances is a rather modest task and was discussed in staggering detail in Chapter 3. In practice however, we need to execute network simulations as part of a system or neuron parameter sweep. Furthermore, due to probabilistic sampling of the test dataset $\mathfrak{D}$ and the input spike trains t^{in}, as well as potentially noisy analogue hardware systems, single experiments must be repeated multiple times. In the optimal case, the outcome of an experiment should always be a distribution of results. Consequently, a rather large number of individual simulation runs is usually performed for each experiment. Especially when investigating small networks, sequential execution of individual simulations is not an option, as the accumulated pre- and post-processing overhead on the hardware systems would be larger than the execution time itself, while large portions of the hardware resources would lie dormant.

To guarantee reproducibility of the test runs, both random data generation and noise sampling must be deterministic – two executions of PyNAM *should execute the exact same network.*

To this end, the Python neural associative memory framework (*PyNAM*) has been developed. It implements the system parameters and evaluation measures presented in Chapter 3, except for input sample pipelining and energy consumption measurements. The tool automatises the process of parameter sweeps and experiment repetition and implements two experiment multiplexing schemes to mitigate the effect of high setup times and to fully utilise available hardware resources. These schemes are *spatial* and *temporal multiplexing*.

For the execution on single-threaded software simulators PyNAM *generates at least as many experiment groups as the number of CPUs in the system, allowing to execute the experiment groups concurrently on all cores.*

Spatial multiplexing is the processes of packing the individual network graphs of independent simulations into a single, concurrently executed graph referred to as *experiment group*. The size of the experiment group is limited by a platform-specific maximum neuron count. While the individual networks in an experiment group are theoretically independent, they may in practice interfere with each other on neuromorphic hardware systems, for example due to congestion of the digital communication network. To mitigate such effects, *PyNAM* randomly shuffles the order in which the individual simulations are executed for each repeated run.

For platforms with global neuron parameters (such as Spikey), PyNAM *ensures equality of these parameters within each experiment group.*

Temporal multiplexing refers to the reuse of the same network graph for a sequence of simulation runs. This is only possible if sweeps over input data encoding parameters are performed (Table 4.1). These solely cause variations in the input datasets, but do not influence the network graph itself. The generated input spike trains corresponding to each individual parameter set are joined into a single compound input spike train by the temporal multiplexing mechanism. Individual parameter sets are separated by a long pause, allowing the neurons to reset. Preceding the analysis, the output spike trains are split into their individual compartments.

The pause between spike trains is chosen as $10 \cdot T$*, where* T *is sample interval of the last experiment.*

As indicated in Figure 5.1, *PyNAM* is separated into three phases: construction and serialisation of the experiment groups according to a JSON experiment description, execution of the experiment groups

An example of the experiment descriptor file is shown in Appendix A.3.

on the software or hardware simulator and the final demultiplexing and analysis resulting in a single HDF5 file [HDF15]. The use of *PyNNLess* as a middleware allows execution of the experiments on all supported platforms. *PyNAM* furthermore automatically adapts the spatial multiplex sizes according the limitations of the current platform.

5.1.3 *Limitations of the hardware platforms*

An overview of the neuromorphic hardware systems and their nominal specifications has been given in Section 2.4. With the exception of Spikey the systems were still in early development at the time of writing. The platforms were neither available in their final size, nor were the software interfaces complete. This results in various limitations regarding the subsequent experiments.

In PyNAM *the maximum number of neurons is limited to* 1 500*, as the excessive use of population objects in* PyNAM *disproportionally slows down the place-and-route algorithm.*

Limitations of NM-MC1 Currently the many core system NM-MC1, as available to HBP researchers, consists of three 48-chip boards theoretically allowing the concurrent simulation of more than 100 000 individual neurons. However, due to limitations in the place-and-route algorithm regarding population objects and how they are used by *PyNAM*, the maximum number of neurons is limited to one per processor core, amounting to about 2100 LIF neurons. Apart from this restriction – which is likely to be resolved in the near future – the system works as intended.

Limitations of NM-PM1 and ESS Unfortunately, the same cannot be said of the physical model NM-PM1. While the network basically runs on the platform, known issues with the current revision of the HICANN-chip cause a rather erratic behaviour of the platform. Due to these issues, no reasonable experiments could be conducted on NM-PM1. An older revision of the NM-PM1 software emulator, the ESS, is used instead. However, due to detailed hardware emulation, the system is far to slow for two-dimensional parameter sweeps.

Limitations of Spikey As already mentioned in Section 2.4.3, the single-chip analogue neuromorphic hardware system Spikey (Figure 1.1 and Section 2.4.3) has rather severe restrictions regarding the neuron parameter space of its 192 LIF neurons: the excitatory reversal potential and membrane capacitance are constant at $E_e = 0\,\mathrm{mV}$ and $C_m = 0.2\,\mathrm{nF}$, and the parameters E_i, E_{Th}, E_L, E_{reset} are shared in blocks of 96 neurons and limited to a maximum of $-55\,\mathrm{mV}$. The parameters g_L and τ_{ref} can be set individually per neuron. The synapse time constants τ_e and τ_i cannot be provided by the user. They implicitly depend on the synaptic weight w and the membrane leak conductivity g_L [Pfe+13].

5.2 NEURON PARAMETER EVALUATION

In Section 4.7.2, we performed two neuron parameter sweep experiments aimed at empirical comparison of the single neuron evaluation measures. However, these experiments can not assess the predictive power of single neuron evaluation for the storage capacity I of entire networks. In this section we aim at bridging this gap by repeating the previous parameter sweeps with full BiNAM networks instead of single neurons. The full network simulations are performed with LIF neurons on the software simulator NEST, and independently on the fully digital NM-MC1 and analogue Spikey hardware systems. This allows both verification of the single neuron evaluation measures and comparison of the hardware performance to the reference software simulator.

Subsequently, we first elaborate on the experiment methodology, followed by a description of the results for NM-MC1 and Spikey. Finally, we discus the insights gained from the parameter sweeps.

5.2.1 *Methodology*

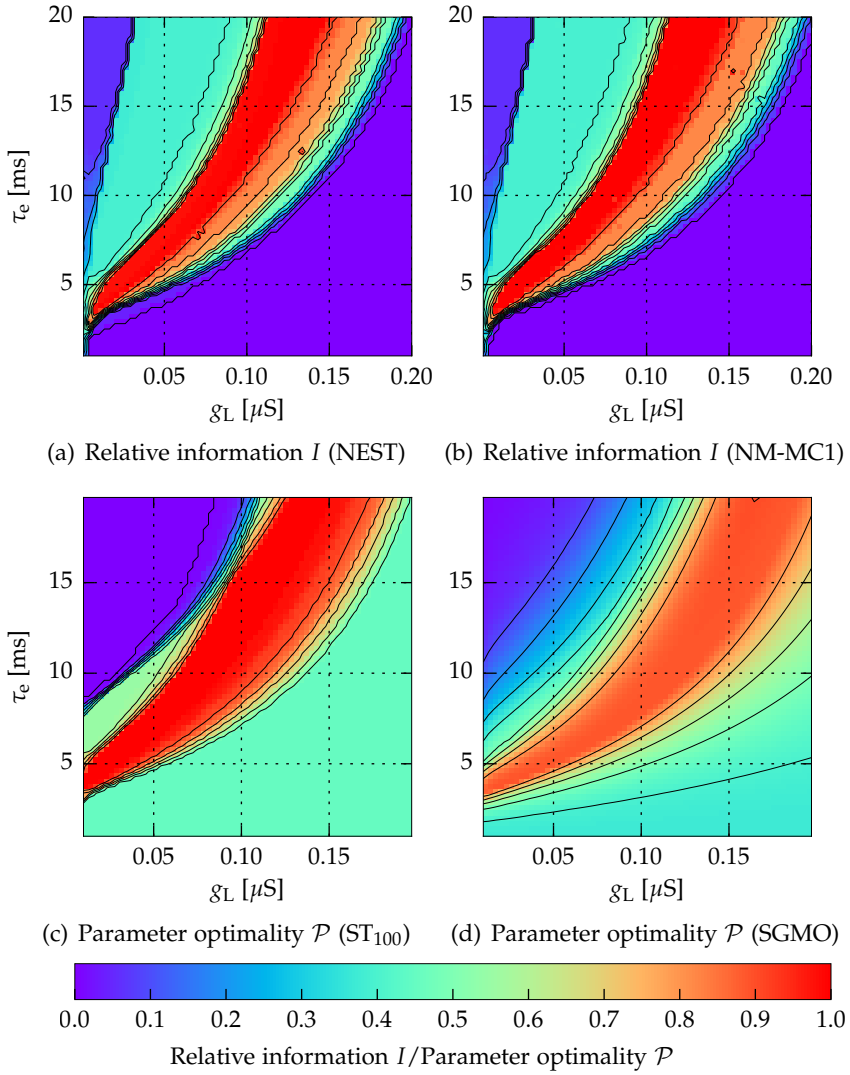
Analogously to the parameter sweeps in Section 4.7.2, the initial neuron parameters are selected according to Table 4.8. System parameters which were already relevant for single neuron evaluation follow *Scenario I* in Table 4.10. Additionally, the network size is selected as $m = 16$ input signals and $n = 16$ output neurons, with $c = 3$ one-bits in the input data vectors $\vec{x}_k$ (as assumed in the single neuron evaluation) and $d = 3$ one-bits in $\vec{y}_k$. According to Equations (2.36) and (3.23) the maximum storage capacity is reached for this data configuration at $N = 27$ input samples. The particular dataset $\mathfrak{D}$ generated by *PyNAM*, achieves a maximum storage capacity of $I = 227\,\text{bit}$ with ten expected false positives over all test samples.

This supposedly small network size was chosen in order to limit both the time required for network generation in PyNAM *and the NEST simulation to sensible ranges.*

The optimal input sample count is automatically selected by PyNAM.

The neuron parameter vectors are sampled from a uniformly spaced 64×64 grid. For each parameter vector, the false positive and negative counts n_{fp}, n_{fn}, the information measure I, and the average latency δ per sample are calculated. Information measure and false negative/positive counts are normalised, to allow direct comparison of the information I with the single neuron evaluation measures and to improve the comparability of full network evaluations based on distinct system parameters. The false positive count n_{fp} is normalised to the range $[-1, 1]$, where $n_{\text{fp}} = -1$ corresponds to "none of the expected false positives are generated", $n_{\text{fp}} = 0$ to "the expected number of false positives was generated" and $n_{\text{fp}} = 1$ to "all bits is in the output were set to one". False negative count and information are expressed relative to the maximum possible false negative count and the maximum information respectively.

The false positive normalisation is only piecewise linear: there are 10 *expected false positives, but* 351 *possible false positives. Correspondingly* $[0, 10]$ *is mapped to* $[-1, 0]$ *and* $[10, 351]$ *to* $[0, 1]$.



(a) Relative information I (NEST) (b) Relative information I (NM-MC1)

(c) Parameter optimality $\mathcal{P}$ (ST_{100}) (d) Parameter optimality $\mathcal{P}$ (SGMO)

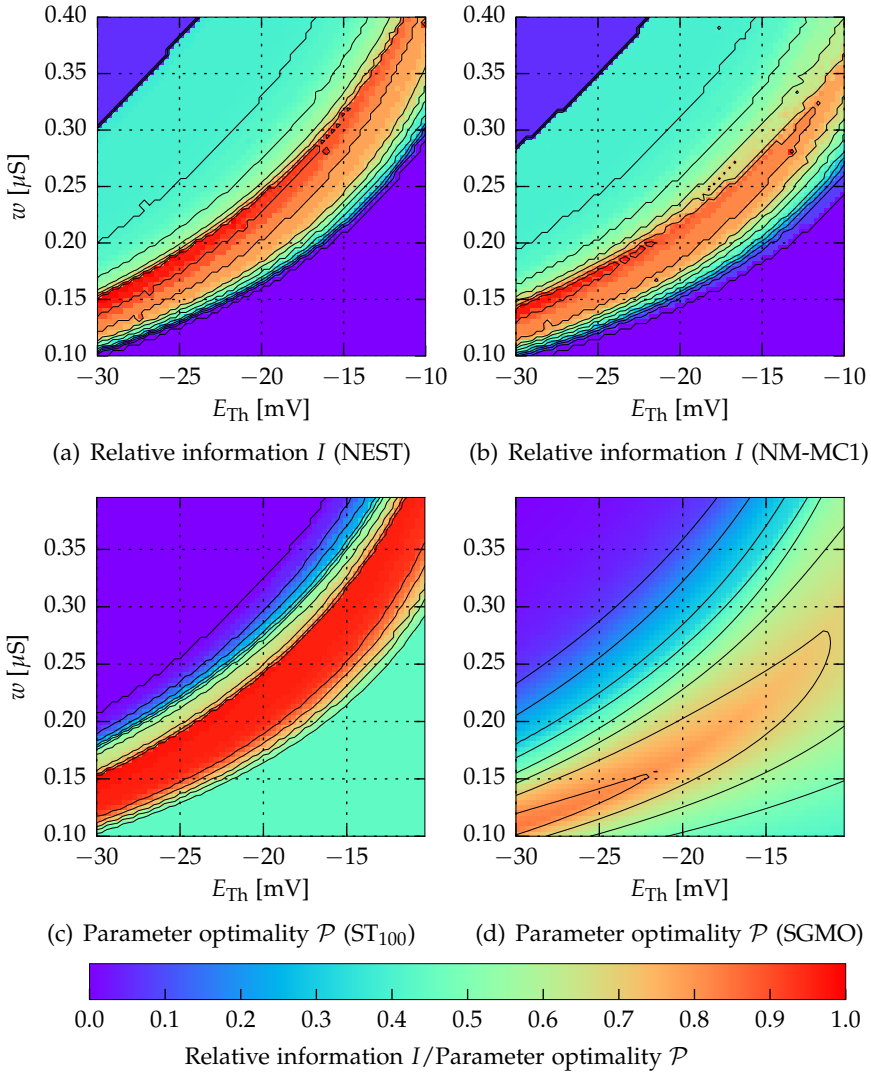
Figure 5.2: Comparison of the g_L/τ_e sweep on the NEST and NM-MC1 platforms with the single neuron evaluation results. Refer to Sections 5.2.1 and 5.2.2 for more information.

5.2.2 *Neuron parameter sweep on NM-MC1*

The results for SGSO are not compared. Interested readers are referred to the corresponding data in Appendix C.

The first parameter sweep executed on NM-MC1 varies g_L from 0.01 µS to 0.2 µS and τ_e from 0 ms to 20 ms. Most strikingly, the results for the NEST software simulation and the NM-MC1 neuromorphic hardware (Figures 5.2(a) and 5.2(b)) are almost identical. The region with high optimality in the ST_{100} single neuron evaluation measure (Figure 5.2(c)) overlaps well with the high-information regions in the full network-evaluation. However, ST_{100} slightly overestimates the parameter quality. The same is true for the SGMO measure (Figure 5.2(d)) which is shifted too far towards larger g_L compared to ST_{100}.

Minor discrepancies between NEST and NM-MC1 can be found in the second parameter sweep, which varies w from 0.01 µS to 0.2 µS and τ_e from 0 ms to 20 ms. As shown in Figure 5.3(a), the region of



(a) Relative information I (NEST)

(b) Relative information I (NM-MC1)

(c) Parameter optimality $\mathcal{P}$ (ST_{100})

(d) Parameter optimality $\mathcal{P}$ (SGMO)

Figure 5.3: Comparison of the E_{Th}/w sweep on the NEST and NM-MC1 platforms with the single neuron evaluation results. Refer to Sections 5.2.1 and 5.2.2 for more information.

maximal information is minimally larger on NEST than on NM-MC1 (Figure 5.3(b)). ST_{100} is again too optimistic in comparison to the ground truth provided by NEST (Figure 5.3(c)). SGMO is shifted towards smaller w, yet exhibits a gradient leading to a global maximum in the region in which NEST shows the highest information.

Comparison of the false positive and false negative counts for the second sweep in Figures 5.4(a) to 5.4(d) shows no striking difference between NEST and NM-MC1 apart from a marginally smaller false positive count for NEST in the region of maximal information. The largest discrepancies between NEST and NM-MC1 can be found in Figures 5.4(e) and 5.4(f), which contain the latency plots. Whereas NEST exhibits a smooth wavy grain pattern, the same pattern is interspersed with noise on NM-MC1.

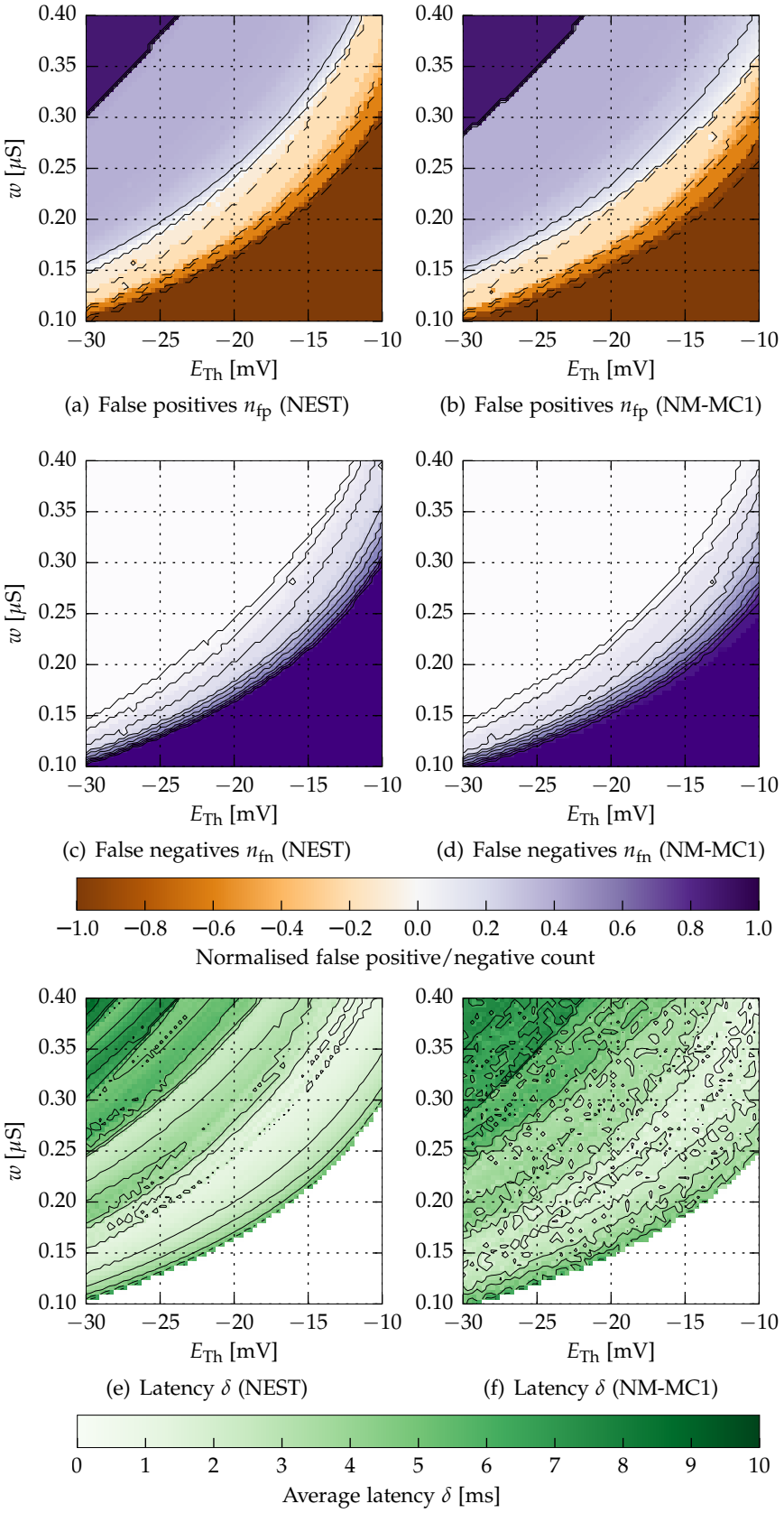


Figure 5.4: Comparison of the latencies, false positive and false negative counts for the E_{Th}/w sweep on NEST and NM-MC1. Refer to Sections 5.2.1 and 5.2.2 for more information.

5.2.3 *Neuron parameter sweep on Spikey*

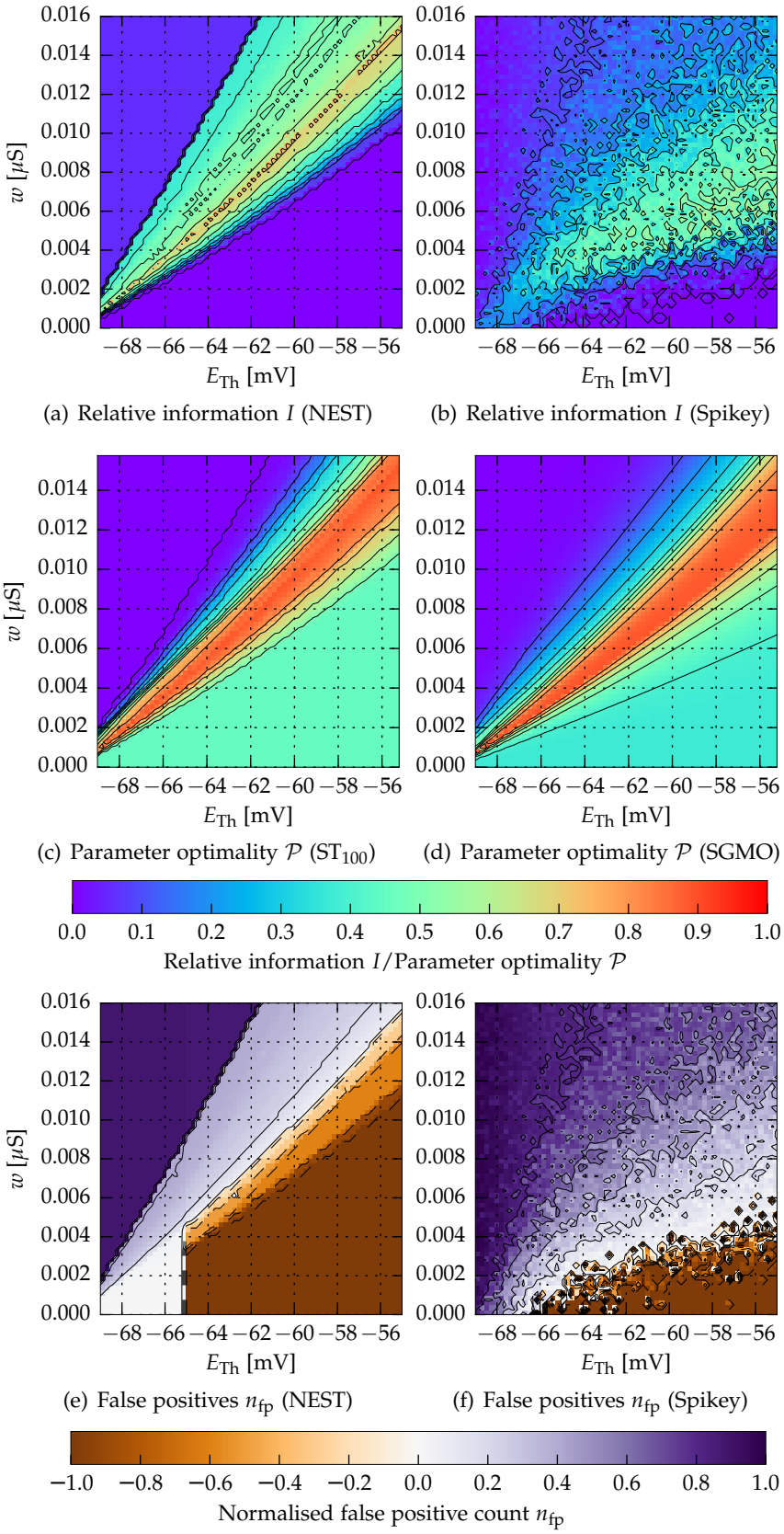
The excitatory channel time constant τ_e is not user-definable on Spikey. Correspondingly, the g_L/τ_e sweep cannot be performed. The threshold potential E_{Th} is furthermore limited to a maximum of $-55\,\text{mV}$, which lies outside the range of the previous sweep. A maximally large range (with respect to E_L at $-70\,\text{mV}$) is chosen instead and E_{Th} is varied from $-69\,\text{mV}$ to $-55\,\text{mV}$. The synaptic channel amplitude is reduced along with E_{Th} by lowering the w-range to $0.0\,\mu\text{S}$ to $0.016\,\mu\text{S}$. Note that the membrane capacitance of Spikey is $C_m = 0.2\,\text{nF}$ instead of the previously used $1.0\,\text{nF}$.

The results of the parameter sweep on Spikey are shown in Figures 5.5 and 5.6. In contrast to the previous experiments, the storage capacity measure differs significantly between the software simulator NEST and the neuromorphic hardware (Figures 5.5(a) and 5.5(b)). In the NEST simulation the region of highest information is spread along a straight line, a behaviour once again reproduced by both ST_{100} and SGMO, though the results for SGMO are skewed towards smaller w (Figures 5.5(c) and 5.5(d)). Both NEST and the single neuron evaluation measures coherently indicate that the theoretical maximum information/optimality cannot be reached for the given parameters. The information graph for Spikey is distorted, extremely noisy and has a smaller maximum information than the NEST simulation.

The comparison of the false positive measure between NEST and Spikey in Figures 5.5(e) and 5.5(f) bears another interesting detail. Whereas a graduated transition from regions with negative n_{fp} (regions with too few output spikes) to regions with positive n_{fp} (regions with too many output spikes) is present in the false positive count for NEST, such a transition is almost completely missing on Spikey. The system either produces no Spikes at all or too many spikes. Furthermore, the latency measure in Figure 5.6 shows severe noise and larger latency for Spikey compared to NEST.

5.2.4 *Discussion*

Above all, the experiments highlight two facts. Firstly, the spiking neural network implementation of the Willshaw associative memory described in Chapter 3 is operational on neuromorphic hardware, albeit the results for Spikey are suboptimal. In contrast, the neuron parameter sweep on NM-MC1 found an entire region in the parameter space which achieves the theoretical maximum storage capacity (Figure 5.2(b)). Secondly, the single neuron evaluations cohere extremely well with the full network storage capacity measure I as calculated by the reference network simulator NEST. This supports the conjecture put forth in Chapter 4: analysis of a single neuron in the network is sufficient to predict the performance of the full network. Nevertheless,



(a) Relative information I (NEST) (b) Relative information I (Spikey)

(c) Parameter optimality $\mathcal{P}$ (ST_{100}) (d) Parameter optimality $\mathcal{P}$ (SGMO)

(e) False positives n_{fp} (NEST) (f) False positives n_{fp} (Spikey)

Figure 5.5: Comparison of the E_{Th}/w sweep on the NEST and Spikey platforms with the single neuron evaluation results and false positive counts. Refer to Sections 5.2.1 and 5.2.3 for more information.

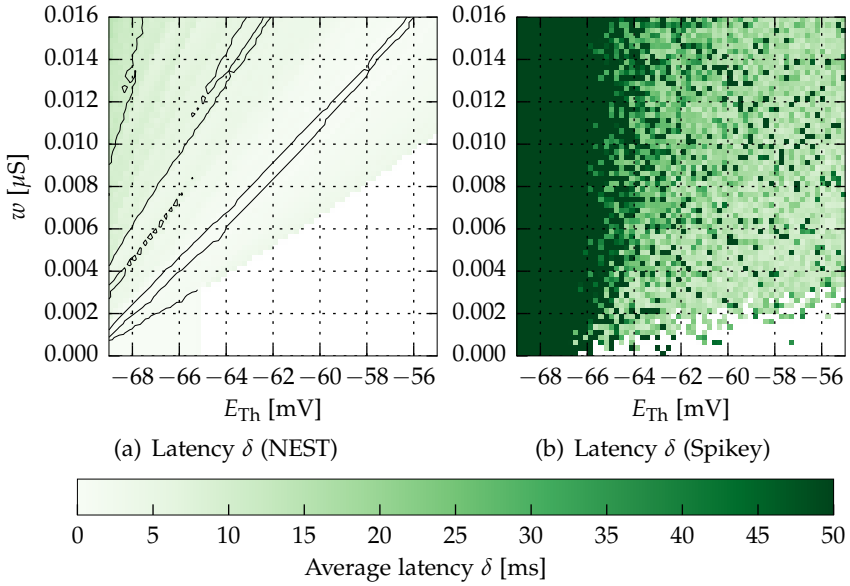


Figure 5.6: Latencies measured during the Spikey parameter sweep in comparison with the NEST simulation. Contour lines were not drawn in (b) for a better overview. Refer to Sections 5.2.1 and 5.2.3 for more information.

as mentioned in the last chapter, further research is required to correct the slightly deviating behaviour of the otherwise promising SGMO measure.

The experiments show that NM-MC1 implements the reference LIF neuron model extremely well. With the results of the differential equation integrator benchmark in Section 4.2.4 in mind, the minor deviations from the reference are well explainable with the constant $h = 1\,\text{ms}$ step size in the NM-MC1 numerical neuron integrator. The fact that all spikes in NM-MC1 are discretised to a one-millisecond grid furthermore explains the noise in the memory latency.

The results for the analogue Spikey system are less fortunate. Even though it is visible that Spikey approximately follows the behaviour of the reference, the result is superimposed with severe noise. Possibly, the use of neuron populations might improve the situation, as these theoretically lessen the impact of single noisy neurons (Section 3.1.3). Nevertheless, the main advantage of analogue neuromorphic hardware must not be kept out of sight. Spikey is extremely fast: execution of the above parameter sweep took about ten minutes on Spikey, forty minutes on NM-MC1 and five hours on NEST, including all network generation and analysis overhead. More complex biologically inspired networks might not be as susceptible to noise as the BiNAM. However, it is exactly this susceptibility to noise or other deviations from the norm which suggest that the BiNAM is a helpful benchmarking network for neuromorphic platforms.

5.3 SYSTEM PARAMETER SWEEPS

While the above two-dimensional neuron parameter sweeps are clearly suitable as a benchmark, their computation is rather time consuming and the resulting three-dimensional graphs are hard to analyse. Of course, an intriguingly simple way of comparing two platforms is to just calculate the storage capacity for a constant neuron and system parameter set. However, these parameters must possess a sufficiently large discriminatory power: they should neither pose a barely solvable nor a too trivial problem. A solution to this dilemma is to perform a system parameter sweep which constructs a series of gradually more difficult tests.

This idea is the basis of the so far not tested robustness measure (Section 3.2.2), which sweeps over an arbitrary noise parameter σ, and the critical time window analysis (Section 3.2.3), which varies the time window T. In this section we analyse these measures with respect to their suitability as hardware benchmark. To this end, exemplary one-dimensional system parameter sweeps are performed on the NEST software simulator, the NM-MC1 digital neuromorphic hardware system, ESS as a replacement for NM-PM1 and Spikey as an analogue neuromorphic hardware system. As before, we begin with a short description of the methodology, followed by the results and a discussion.

5.3.1 *Methodology*

The one dimensional system parameter sweeps performed in this section measure the storage capacity I over a system parameter. The initial system and neuron parameters are those already employed in the previous experiments. Their description can be found in Section 5.2.1.

To ensure a maximal value range for I, the neuron parameters Φ must be optimised prior to the experiment with respect to the initial system parameters. To bear any meaning as a benchmark, the optimisation process targets the reference platform (NEST), and the neuron parameters Φ are shared across all platforms. To this end, the Spikey neuron parameter space, a strict subset of the parameter spaces of the other platforms, must be used in the neuron parameter selection. The LIF neuron membrane capacitance and threshold potential are correspondingly set to $C_\mathrm{m} = 0.2\,\mathrm{nF}$ and $E_\mathrm{Th} = -55\,\mathrm{mV}$. According to the neuron parameter sweep in Figure 5.5(a), a maximum in the storage capacity is reached for $w = 16\,\mathrm{nS}$. A preliminary test run on NEST shows that these parameters reach the theoretical maximum storage capacity if the spike time jitter is reduced from $\sigma_t = 5\,\mathrm{ms}$ to $2\,\mathrm{ms}$ in the initial system parameters.

In the following, the system parameters σ_t, σ_w and T are linearly sampled from a given value range in fifty steps. For each sample, the

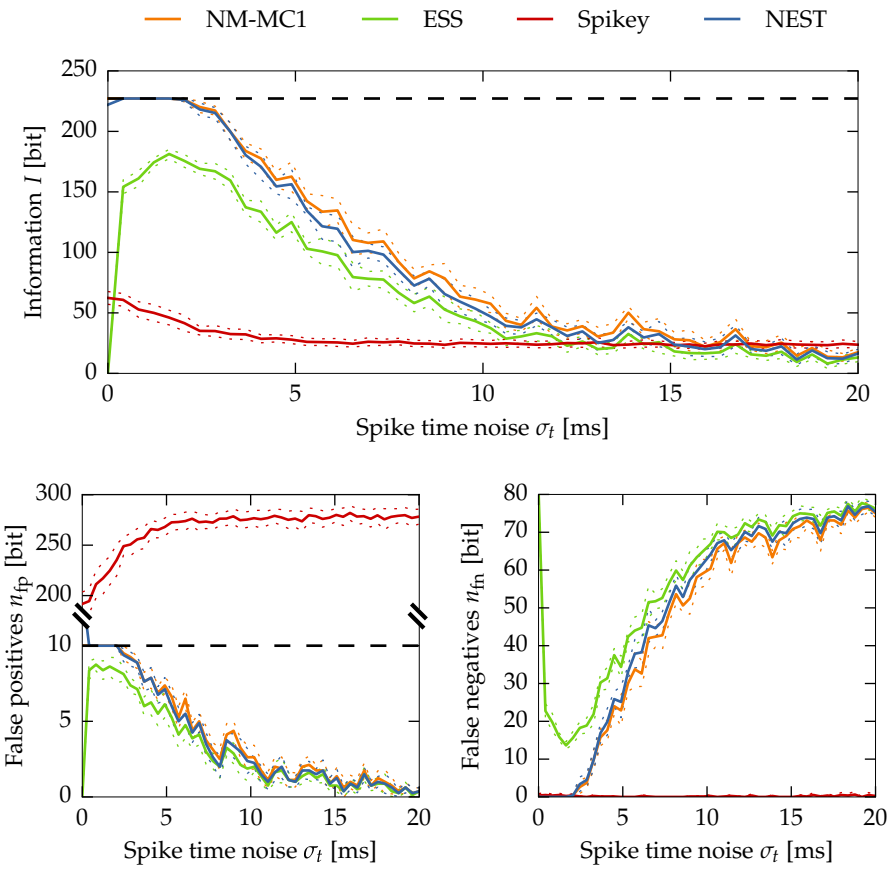


Figure 5.7: Results of the spike time noise parameter sweep. Shown are the information and the corresponding false positive/false negative counts. Solid lines correspond to the mean over eight runs, dotted lines to the $\pm 0.5 \cdot \sigma$ standard deviation. The dashed lines show the maximum information and the expected false positive count respectively.

memory storage capacity I is measured in eight independent runs, over which mean and standard deviation of I are calculated.

5.3.2 *Experimental results*

In the first experiment, the standard deviation of the Gaussian spike time noise σ_t is varied from 0 ms to 20 ms. The results are shown in Figure 5.7. As expected, the information measure decreases with increasing σ_t. NEST and NM-MC1 exhibit a very similar behaviour: both start with the theoretical maximum information and slowly decrease towards zero information. Interestingly, ESS begins with zero information at $\sigma_t = 0$ ms and rapidly reaches a relatively large yet not maximal I at $\sigma_t = 2$ ms. It then converges towards the result for NEST and NM-MC1. Spikey starts with a small I and converges to a value which is slightly larger than the final result of the other measures.

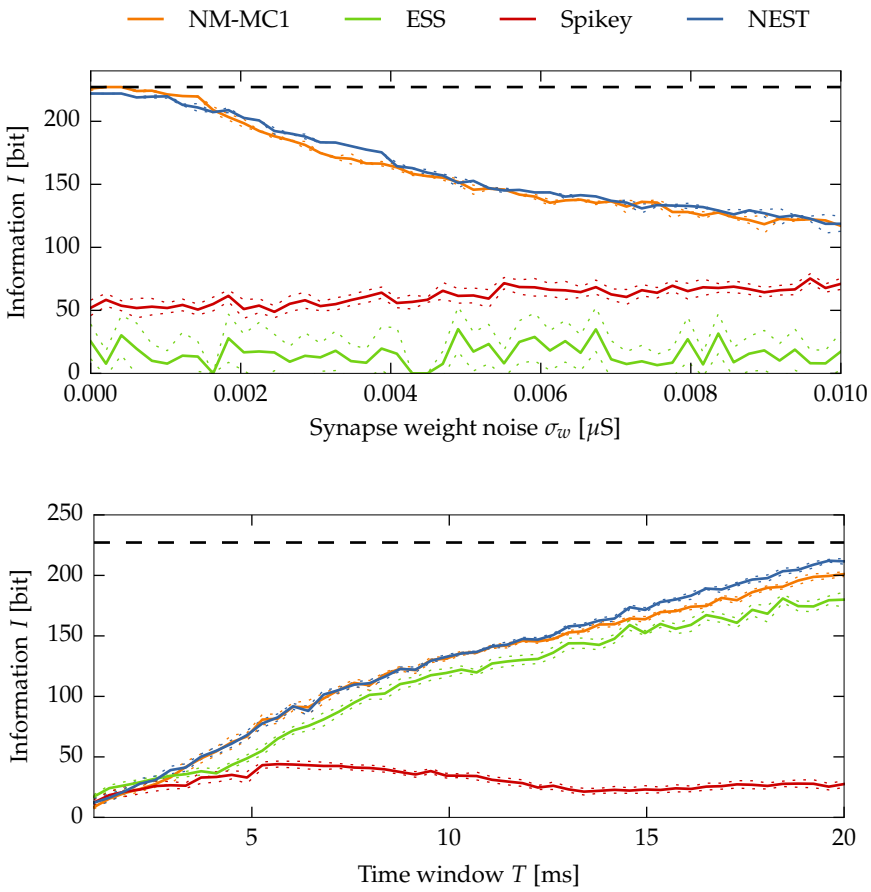


Figure 5.8: Results of the spike time noise and time window sweeps showing the information I over the system parameters. Solid lines correspond to the mean over eight runs, dotted lines to the $\pm 0.5 \cdot \sigma$ standard deviation. The dashed line shows the maximum information.

Comparison of the false positive and false negative counts shows that NEST, NM-MC1 and ESS exhibit a similar overall behaviour with a decreasing false positive count and an increasing false negative count. The zero information at $\sigma_t = 0\,\text{ms}$ for ESS is caused by a maximally large initial false negative count $n_{\text{fn}} = 81$. Spikey exhibits a different behaviour, with an increasing false positive count and a constant zero false negative count.

Figure 5.7 shows the information measure as a result of the sweep over the synapse weight noise σ_w (from 0.0 nS to 10.0 nS) and the time window T (from 2 ms to 20 ms). Unsurprisingly, the information measure decreases with increasing synapse weight noise for NEST and NM-MC1. Surprisingly, ESS exhibits very small values over the entire sweep with a large standard deviation. For Spikey, the information starts at a small value and unexpectedly increases slightly with larger noise values.

The time window analysis exhibits the behaviour that was anticipated in Section 3.2.3, with a coherent decrease in the information for

smaller time windows T. Systematic differences in the performances of the platforms are clearly visible.

5.3.3 *Discussion*

Just like the two-dimensional neuron parameter sweeps, the one dimensional system parameter sweeps are a valuable benchmark. They show a clear separation between models with reference behaviour on the one hand (NEST and NM-MC1), and Spikey and ESS on the other hand. In two of the sweeps the performance exhibited by ESS is close to the performance of NEST and NM-MC1, and significantly better than the erratic performance displayed by Spikey, which is – as in the previous experiment – characterised by a tremendously large false positive count.

Furthermore, the sweeps uncover suboptimal and interesting behaviour of the platforms. Strikingly and for unknown reasons, ever so slight variations in the synaptic weight cause a breakdown of ESS performance. The same is true for zero spike time noise. However, a plausible explanation for this behaviour exists: the platform seems to fuse input spikes arriving at the exact same time. Since all spikes arrive at the same time for $\sigma_t = 0\,\text{ms}$, the neuron produces no output spikes and thus exhibits the large false negative count.

However, the results also show that care has to be taken regarding the range of the sweep. In case platforms show an erratic behaviour they may achieve a larger I as platforms with a systematic behaviour, as it is the case for the for large σ_t, for which Spikey exhibits a slightly larger I than the other platforms.

5.4 CONCLUSION

In this chapter we have presented the software pipeline for full network evaluation. This pipeline has been employed to conduct two sets of experiments on both neuromorphic hardware and in software simulation. The first set of experiments confirmed the predictive power of single neuron evaluation, found a high accordance between the results from NM-MC1 and NEST, and a certain dissonance between the results for NEST and Spikey. The second experiment was similarly capable of discriminating between the individual simulation platforms. As such, the BiNAM is well suited as a benchmarking network.

Most important, yet almost unmentioned, is the fact, that the primary goal of this thesis – the implementation of an operational Willshaw associative memory as spiking neural network on neuromorphic hardware – has been reached. Correspondingly, we can now proceed to the final chapter.

6 CONCLUSION AND OUTLOOK

In this final chapter we summarise the original work presented in the previous chapters as well as the insights gained from the experiments. The summary is followed by a catalogue of possible future work and the final conclusion.

6.1 SUMMARY

In Chapters 3 and 4, an entire pipeline for construction, execution and evaluation of spiking implementations of the Willshaw associative memory (BiNAM) has been described. This pipeline includes the selection of the network topology, data encoding and appropriate neuron parameters (Section 3.1 and Chapter 4), the generation of memory test data (Section 3.3) and measures for the assessment of the memory performance (Section 3.2).

Neuron parameter selection itself has been based on three single neuron evaluation measures, discussed in Sections 4.3 to 4.5. The SGMO measure employs the novel *fractional spike count* technique to estimate a smooth, computationally efficient network performance prediction, which is – despite pending issues (Sections 4.7.3 and 5.2) – deemed suitable for parameter exploration and optimisation.

A comprehensive software collection accompanies this thesis. With *AdExpSim*, a framework for high performance single neuron simulation has been developed. Among other applications, it provides a graphical tool for interactive design space exploration (Section 4.6). The *PyNNLess* library allows simple, platform agnostic simulation of spiking neural networks in Python (Section 5.1.1). Built on top of *PyNNLess* is *PyNAM*, which conducts configurable parameter sweeps over the entire design space by simulating full BiNAM networks on any selected target platform (Section 5.1.2).

In Chapter 5, series of experiments on neuromorphic hardware systems and software simulators have been described, which conclusively show that the presented pipeline indeed allows the construction of an operational Willshaw associative memory. Full network design space explorations cohere well with the single neuron evaluation measures, underpinning their predictive power. Furthermore, one dimensional system parameter sweeps highlight significant differences in the results for the individual hardware platforms, confirming the conjecture that the BiNAM is suitable as a low-level benchmark for neuromorphic hardware systems.

6.2 FUTURE WORK

For the time being, some of the threads taken up in this thesis could not be spun to an end. Instead, a multitude of research topics has emerged from the presented work. This section lists both important tasks in the context of HBP and lesser, yet interesting, future work.

6.2.1 *Large scale simulations and benchmarking*

The most obvious open task is the systematic large scale simulation and benchmarking of the BiNAM on neuromorphic hardware. The feasibility of this task mainly depends on the availability and stability of the neuromorphic hardware and its associated software stack. It will be especially interesting to repeat the experiments on the new HICANN chip revision for the NM-PM1 system. Large scale networks encompassing thousands of neurons could additionally uncover scaling issues in the hardware systems.

A proposed solution to the first software-wise challenge is the development of a dedicated network generator in C++. For the second problem, the PyNAM *tool must consolidate neurons in the BiNAM into larger population objects, instead of issuing individual populations for each neuron. This was not possible at the time of writing due to issues with the hardware bindings.*

Regarding the software presented in this thesis, two open challenges must be overcome. While it is fully functional, the performance of the BiNAM network generator implemented in *PyNAM* is a major bottleneck and has to be improved. Generating parameter sweeps for large networks may take multiple hours, in contrast to potentially much shorter run times on neuromorphic hardware. Secondly, the capabilities of NM-MC1 are currently not utilised to the fullest. Further optimisation is needed in both cases.

6.2.2 *Neglected design space parameters*

Some of the full network evaluation measures and design space parameters proposed in Chapter 3 could not be tested. This includes the energy consumption, the impact of neuron populations, as well as input sample pipelining. Verification of the energy prediction in Equation (3.12) requires access to the ground truth data collected by the neuromorphic platforms, which is not yet available. It would be interesting to see whether the use of neuron populations improves the performance of the networks as anticipated. Implementation of the input data pipelining scheme presented in Section 3.1.1 would allow to operate the network at its limits and further improve the discriminatory properties of the network as a benchmark.

6.2.3 *Extensions of the BiNAM network*

The Willshaw associative memory model was implemented in this thesis in its most fundamental form. Multiple extensions of the model exist, which will be interesting to explore once the basic memory has

been verified to work as intended on all hardware platforms. A first extension is the implementation of an adaptive threshold (Section 2.5.6), which would allow practical applications inside larger networks, such as the pattern completion example in Section 2.5. Another possible extension is online training of synaptic weights using Spike-timing dependent plasticity (STDP), a learning mechanism implemented on the HBP neuromorphic hardware systems [SG10]. Finally, the use of inhibitory synapses and more advanced models such as the spike-counter model could be studied [Kno03; Kno+14; Mül15].

6.2.4 *Neuron evaluation and parameter optimisation*

The SGSO and SGMO single neuron evaluation measures do not perfectly cohere with the ground truth provided by the "spike train" measure. As elaborated in Section 4.7.2, this is most likely an effect of the multiplication of the individual terms $p_1, p_0, p_{\text{reset}}$, which overlap in a suboptimal way. As a solution, parameter tuning (slope of the sigmoid and bell-shaped distributions) or more intelligent combination of terms may be explored.

Both, an implementation of an adaptive step size integrator and the SGMO measure could be more challenging on a GPU due to incoherent branching.

A GPU implementation of the two-dimensional visualisation of the single neuron evaluation measures in *AdExpSim* is another possible goal. The problem is intrinsically parallelisable, since each grid point in the visualisation corresponds to an independent single neuron evaluation. At least for the spike train and SGSO measure, negligible memory usage and coherent branching are perfect prospects for a GPU implementation.

Regarding parameter optimisation, less naive methods than the Downhill Simplex with potentially faster convergence should be tested. Another extension to the optimisation process is closed-loop operation in conjunction with the neuromorphic hardware, which would eliminate any possible discrepancy between idealised single neuron simulation and the actual behaviour of the target platform.

6.2.5 *Fractional spike count measure*

The fractional spike count measure is a promising tool for general purpose neuron parameter optimisation, a problem commonly considered hard to solve. At least for applications in which neuron parameters are optimised with respect to a certain output spike count for a constant input, this method could complement current approaches [Pri07]. It would be interesting to see whether anomalies and computational complexity of the fractional spike count measure can be further reduced, for example by incorporating information about the neuron dynamics. Since the method is largely neuron model agnostic, it should be tested with more complex models, such as the Hodgkin-Huxley (HH) model [HH52].

6.3 CONCLUSION

The thesis has reached its primary goals to describe the design space of a spiking neural network implementation of the Willshaw associative memory, to provide the tools necessary for design space exploration and to successfully execute the memory on neuromorphic hardware.

In order to benefit from the comprehensive theoretical framework encompassing the memory, the initial decision has been made to implement the network as close to the Willshaw model as possible, which in return necessitates careful tuning of the neuron parameters. Neuron parameter selection has been discussed at great length, starting with the neuron models themselves and their efficient numerical simulation, and finally culminating in three distinct evaluation measures. These have been shown to predict the behaviour of the full network. Equipped with the theoretical foundation of the Willshaw associative memory and optimised neuron parameters, the properties of the memory as a hardware benchmark have been examined and found to be highly promising. The presented work is a starting point for further research into associative memories in the context of the Humain Brain Project, and a yet so small step towards the development of artificial cognitive systems.

A CODE EXAMPLES

A.1 SINGLE NEURON INTEGRATOR INTERFACE

As discussed in Section 4.6.3, the single neuron simulator must be highly flexible and at the same time as fast as possible. However, some of the demanded features are intrinsically slow, as they either require access to large chunks of memory (recording) or introduce additional branch instructions (early abortion), while the neuron integrator itself consists of few branches and solely accesses the neuron state vector $\vec{v}$, which fits into a single vector register. The best solution to this features-versus-performance dilemma is to reduce each individual simulator instance to the actually required code. This is achieved by using the C++ template mechanism. The simulator function in the core library is declared as follows:

```
template <uint8_t Flags = 0, typename Recorder,
    typename Integrator, typename Controller>
static void simulate(const SpikeVec &input, Recorder &recorder,
    Controller &controller, Integrator &integrator,
    const Parameters &p, Time tDelta, Time tEnd = MAX_TIME,
    const State &v0 = State())
```

The `Flags` template parameter is a bit-mask used to deactivate parts of the neuron model (degrading it to LIF, deactivating spike handling), or to enable special features, such as support for perturbations. Since template parameters are evaluated at compile time, the compiler strips superfluous code paths from the individual simulator instances. The `recorder`, `controller` and `integrator` arguments are duck-typed, allowing references to objects of any type to be passed as long as the instantiation of `simulate` can be compiled [Str13]. This allows to implement any kind of recording, cancellation (through the `controller`) and integration behaviour, without imposing overhead on other instances of the simulator. The *AdExpSim* library comes with a variety of predefined recorders, controllers and integrators. The following code snippet demonstrates their usage:

```
const Parameters params; // < Use default parameters
DefaultController controller; // < Wait for neuron to settle
DormandPrinceIntegrator integrator; // < Adaptive integrator
/* Record the neuron parameter traces and print them */
CsvRecorder<> recorder(params, 0.1e-3_s, std::cout);
/* Single LIF neuron with deactivated spiking and input spikes
   at 5 and 10 ms */
Model::simulate<Model::IF_COND_EXP | Model::DISABLE_SPIKING>(
        {{5_ms}, {10_ms}}, recorder, controller,
        integrator, params, Time(-1));
```

A.2 PYNNLESS CODE EXAMPLE

The following code example demonstrates the usage of the *PyNNLess* library. It creates a single LIF neuron, a spike source, connects both, and records a membrane potential trace.

```
import pynnless as pynl
params = {
    "cm": 0.2,        "v_reset": -70,
    "e_rev_E": -40,   "v_thresh": -47,
    "e_rev_I": -60,   "tau_m": 409.0,
    "v_rest": -50,    "tau_refrac": 20.0
}
sim = pynl.PyNNLess("spikey") # Open the "spikey" device
res = sim.run(pynl.Network() # Construct the network and run it
        .add_source(spike_times=[0, 1000, 2000])
        .add_population(
            pynl.IfCondExpPopulation(params=params)
                .record_spikes()
                .record_v()
        )
        .add_connection((0, 0), (1, 0), weight=0.024))
for i in xrange(len(res[1]["v_t"])): # Print the trace
    print(str(res[1]["v_t"][i]) + ";" + res[1]["v"][0][i]))
```

A.3 PYNAM EXPERIMENT DESCRIPTOR

This excerpt shows an example of the *PyNAM* experiment descriptor, which performs a two dimensional sweep over g_L and τ_e.

```
{"data": {
  "n_bits_in": 16, /* m */  "n_bits_out": 16, /* n */
  "n_ones_in": 3,  /* c */  "n_ones_out": 3,  /* d */ },
"topology": {
  "params": { /* Neuron parameters */ },
  "neuron_type": "IF_cond_exp",
  "w": 0.03, "multiplicity": 1 /* K */},
"input": {
  "burst_size": 1, /* sIn */, "time_window": 200.0, /* T */
  "isi": 2.0,  /* Delta T */, "sigma_t": 5.0,
  "sigma_t_offs": 0.0, "p0": 0.0, "p1": 0.0},
"output": { "burst_size": 1 }, // sOut
"experiments": [
  {
    "name": "Sweep gL, tauE",
    "sweeps": {
      "topology.params.g_leak": {
         "min": 0.001, "max": 0.2, "count": 64},
      "topology.params.tau_syn_E": {
         "min": 1.0, "max": 20.0, "count": 64}},
    "repeat": 1
}]}
```

B TABLES

This appendix contains a wide selection of tables which would have taken too much space in the main part.

B.1 RUNGE-KUTTA COEFFICIENTS

The following two tables contain the Runge-Kutta coefficients for three basic, non adaptive differential equation integrators, and the more advanced adaptive Dormand-Prince integrator.

Table B.1: Constant step size integrator Runge-Kutta coefficients for the first-order Euler method, the second-order midpoint method and the classical fourth-order Runge-Kutta method.

(a) Euler

	b_j	1
		1

		ℓ
	$a_{j\ell}$	1
j	1	/

(b) Midpoint

	b_j	1	2
		0	1

		ℓ	
	$a_{j\ell}$	1	2
j	1	/	/
	2	1/2	/

(c) Classical Runge-Kutta

	b_i	1	2	3	4
		1/6	1/3	1/3	1/6

		ℓ			
	$a_{j\ell}$	1	2	3	4
j	1	/	/	/	/
	2	1/2	/	/	/
	3	0	1/2	/	/
	4	0	0	1	/

Table B.2: Runge-Kutta coefficients for the fifth-order Dormand-Prince integrator, which can be used as adaptive step size integrator [DP80].

	b_i	1	2	3	4	5	6	7
		$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0

		ℓ						
	$a_{j\ell}$	1	2	3	4	5	6	7
j	1	/	/	/	/	/	/	/
	2	$\frac{1}{5}$	/	/	/	/	/	/
	3	$\frac{3}{40}$	$\frac{9}{40}$	/	/	/	/	/
	4	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$	/	/	/	/
	5	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$	/	/	/
	6	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$	/	/
	7	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	/

B.2 INTEGRATOR RUNTIME PROFILES

Tables B.3 and B.4 show the relative runtime of non-kernel functions in the differential equation integrator benchmark program with and without exponential function approximation. Functions with a relative runtime smaller than 1% were discarded. Measurements were performed with the sampling profiler `perf`[1] developed as part of the Linux kernel project [Mel10]. The experiment was performed on an Intel Core2Duo E7300 CPU clocked at 2.66GHz. The "○"-module refers to the `AdExpIntegratorBenchmark` main program. See the next section for absolute times.

Table B.3: Single neuron simulator runtime profile with IEEE 754 exponential.

Time	*Module*	*Function*
26.47%	`libm-2.22`	`__ieee754_exp_sse2`
15.24%	○	`BenchmarkResult::compare`
13.97%	○	`Model::simulate<DormandPrinceIntegrator>`
11.05%	`libc-2.22`	`__memmove_ssse3`
6.56%	○	`RungeKuttaIntegrator::integrate`
4.18%	○	`benchmarkSimple<MidpointIntegrator>`
3.97%	○	`std::vector::emplace_back<double>`
3.23%	○	`benchmarkSimple<EulerIntegrator>`
2.51%	○	`Model::simulate<RungeKuttaIntegrator>`

Table B.4: Single neuron simulator runtime profile with approximated exponential.

Time	*Module*	*Function*
20.06%	○	`Model::simulate<DormandPrinceIntegrator>`
17.18%	○	`BenchmarkResult::compare`
14.05%	`libc-2.22`	`__memmove_ssse3`
7.30%	○	`benchmarkSimple<MidpointIntegrator>`
7.21%	○	`RungeKuttaIntegrator::integrate`
5.26%	○	`benchmarkSimple<EulerIntegrator>`
5.01%	○	`Model::aux`
4.54%	○	`std::vector::emplace_back<double>`
3.41%	○	`Model::simulate<RungeKuttaIntegrator>`

B.3 INTEGRATOR BENCHMARK

Tables B.5 to B.7 show absolute time and error values for the AdEx and LIF neuron model. The experiment is repeated for the AdEx model with exponential function approximation. The experiments were performed on an Intel Xeon CPU E5-2620 clocked at 2.00GHz.

1 More information about `perf` can be found at `https://perf.wiki.kernel.org/`.

Table B.5: Differential equation integrator performance for the AdExp model. Errors are RMSE values compared to a Runge-Kutta integration with 100 ns sample time. Percentages represent RMSE normalised to the value range. Simulated time is 10 s at 100 input spike groups with 33 output spikes.

Integrator		*Time and samples*			*Error (RMSE)*								
		t [ms]	N	$\frac{t}{N}$ [µs]	u [mV]	(%)	g_e [nS]	(%)	g_i [nS]	(%)	w [nA]	(%)	Avg. %
Euler	$h = 1$ µs	2476.087	10011577	0.247	0.360	(0.36)	0.001	(0.00)	0.000	(0.00)	0.001	(0.33)	0.17
	$h = 10$ µs	209.023	1002278	0.209	0.932	(0.93)	0.012	(0.01)	0.004	(0.01)	0.001	(0.72)	0.42
	$h = 100$ µs	18.174	101255	0.179	3.530	(3.53)	0.122	(0.11)	0.039	(0.07)	0.011	(6.60)	2.58
	$h = 1$ ms	2.034	11190	0.182	8.358	(8.36)	1.225	(1.07)	0.382	(0.64)	0.017	(10.66)	5.18
Midpoint	$h = 1$ µs	3609.626	10011577	0.361	0.303	(0.30)	0.000	(0.00)	0.000	(0.00)	0.000	(0.29)	0.15
	$h = 10$ µs	329.711	1002278	0.329	0.882	(0.88)	0.000	(0.00)	0.000	(0.00)	0.001	(0.69)	0.39
	$h = 100$ µs	29.946	101249	0.296	3.500	(3.50)	0.001	(0.00)	0.000	(0.00)	0.033	(20.38)	5.97
	$h = 1$ ms	3.047	11161	0.273	42.301	(42.30)	0.088	(0.08)	0.027	(0.05)	2.428	(1516.89)	389.83
Runge-Kutta	$h = 1$ µs	5801.178	10011577	0.579	0.296	(0.30)	0.000	(0.00)	0.000	(0.00)	0.000	(0.29)	0.15
	$h = 10$ µs	542.762	1002278	0.542	0.868	(0.87)	0.000	(0.00)	0.000	(0.00)	0.001	(0.72)	0.40
	$h = 100$ µs	56.256	101250	0.556	4.052	(4.05)	0.000	(0.00)	0.000	(0.00)	0.043	(26.95)	7.75
	$h = 1$ ms	5.410	11149	0.485	48.260	(48.26)	0.000	(0.00)	0.000	(0.00)	2.754	(1720.35)	442.15
Dormand-Prince	$e = 1$ µ	3244.054	1934005	1.677	0.295	(0.30)	0.000	(0.00)	0.000	(0.00)	0.000	(0.30)	0.15
	$e = 10$ µ	774.668	699606	1.107	0.290	(0.29)	0.000	(0.00)	0.000	(0.00)	0.000	(0.28)	0.14
	$e = 100$ µ	80.463	74549	1.079	0.328	(0.33)	0.000	(0.00)	0.000	(0.00)	0.001	(0.32)	0.16
	$e = 1$ m	11.903	10783	1.104	0.573	(0.57)	0.000	(0.00)	0.000	(0.00)	0.001	(0.63)	0.30
	$e = 10$ m	5.444	4549	1.197	1.517	(1.52)	0.003	(0.00)	0.001	(0.00)	0.007	(4.21)	1.43
	$e = 100$ m	4.791	3934	1.218	5.767	(5.77)	0.029	(0.03)	0.008	(0.01)	0.029	(18.18)	6.00

Table B.6: Differential equation integrator performance for the IfCondExp model. Errors are RMSE values compared to a Runge-Kutta integration with 100 ns sample time. Percentages represent RMSE normalised to the value range. Simulated time is 10 s at 100 input spike groups with 33 output spikes.

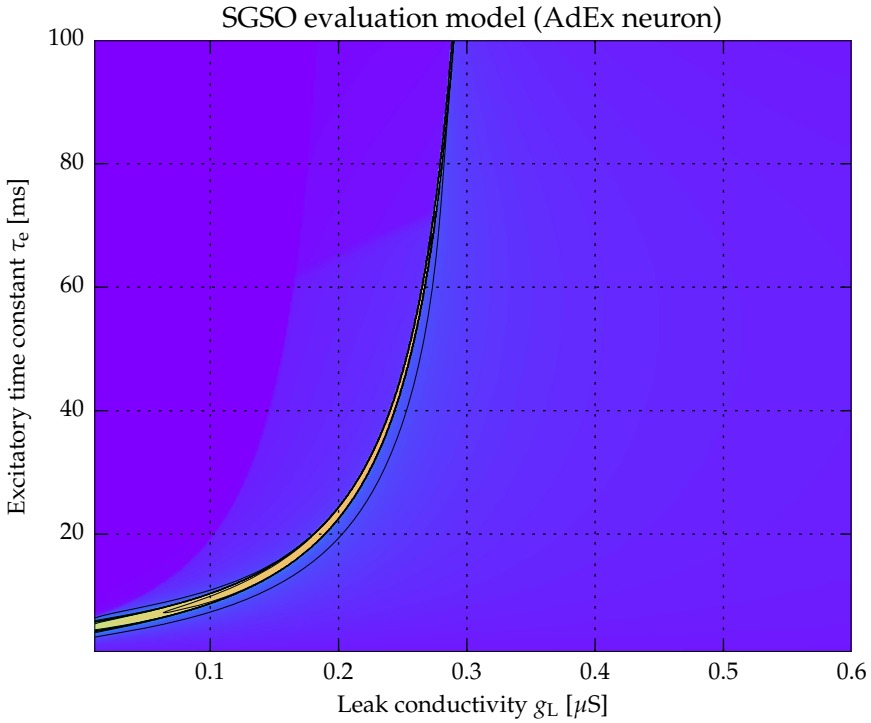
Integrator		*Time and samples*			*Error (RMSE)*								
		t [ms]	N	$\frac{t}{N}$ [µs]	u [mV]	(%)	g_e [nS]	(%)	g_i [nS]	(%)	w [nA]	(%)	Avg. %
Euler	$h = 1$ µs	2071.354	10011577	0.207	0.223	(0.22)	0.001	(0.00)	0.000	(0.00)	0.000	(0.00)	0.06
	$h = 10$ µs	130.804	1002322	0.131	0.739	(0.74)	0.012	(0.01)	0.004	(0.01)	0.000	(0.00)	0.19
	$h = 100$ µs	12.139	101298	0.120	2.508	(2.51)	0.122	(0.11)	0.039	(0.07)	0.000	(0.00)	0.67
	$h = 1$ ms	1.198	11226	0.107	8.079	(8.08)	1.225	(1.07)	0.382	(0.64)	0.000	(0.00)	2.45
Midpoint	$h = 1$ µs	2412.932	10011577	0.241	0.033	(0.03)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.01
	$h = 10$ µs	165.374	1002322	0.165	0.948	(0.95)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.24
	$h = 100$ µs	13.282	101298	0.131	3.156	(3.16)	0.001	(0.00)	0.000	(0.00)	0.000	(0.00)	0.79
	$h = 1$ ms	1.482	11226	0.132	9.820	(9.82)	0.088	(0.08)	0.027	(0.05)	0.000	(0.00)	2.49
Runge-Kutta	$h = 1$ µs	3873.598	10011577	0.387	0.033	(0.03)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.01
	$h = 10$ µs	314.017	1002322	0.313	0.948	(0.95)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.24
	$h = 100$ µs	32.832	101298	0.324	3.155	(3.16)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.79
	$h = 1$ ms	3.024	11226	0.269	9.817	(9.82)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	2.45
Dormand-Prince	$e = 1$ µ	2286.730	1933179	1.183	0.033	(0.03)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.01
	$e = 10$ µ	572.327	699514	0.818	0.336	(0.34)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.08
	$e = 100$ µ	54.232	74227	0.731	1.330	(1.33)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	0.33
	$e = 1$ m	7.367	10398	0.708	4.200	(4.20)	0.000	(0.00)	0.000	(0.00)	0.000	(0.00)	1.05
	$e = 10$ m	2.621	4072	0.644	10.879	(10.88)	0.003	(0.00)	0.001	(0.00)	0.000	(0.00)	2.72
	$e = 100$ m	3.019	3741	0.807	13.645	(13.64)	0.028	(0.02)	0.007	(0.01)	0.000	(0.00)	3.42

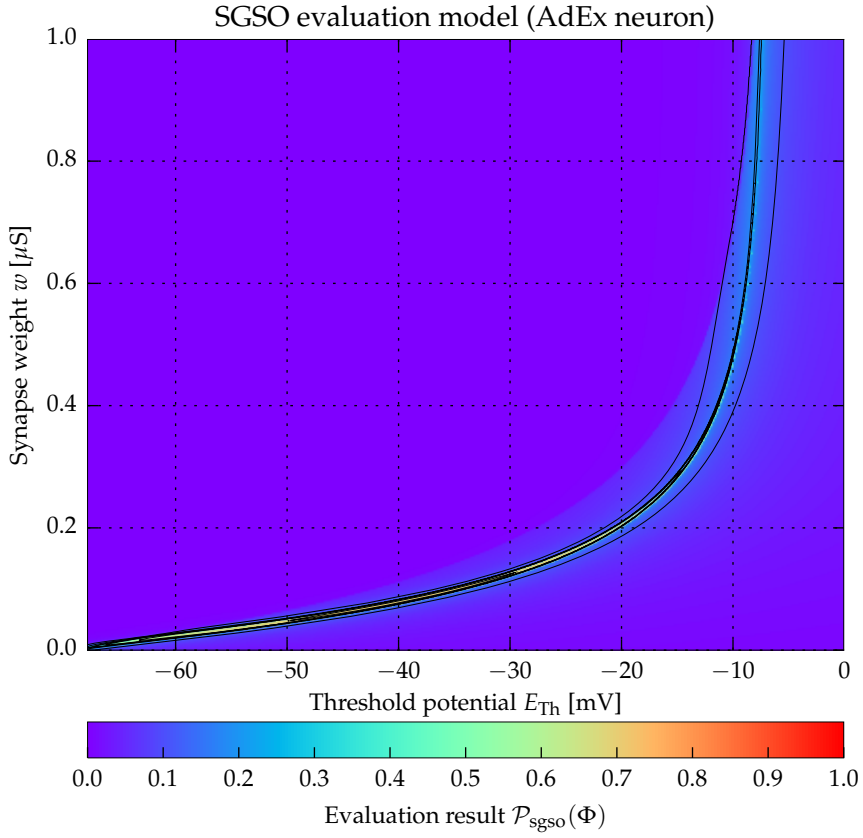
Table B.7: Differential equation integrator performance for the AdExp model with approximation of the exponential function used in the spiking mechanism. Errors are RMSE values compared to a Runge-Kutta integration with 100 ns sample time (and full precision exponential). Percentages represent RMSE normalised to the value range. Simulated time is 10 s at 100 input spike groups with 33 output spikes.

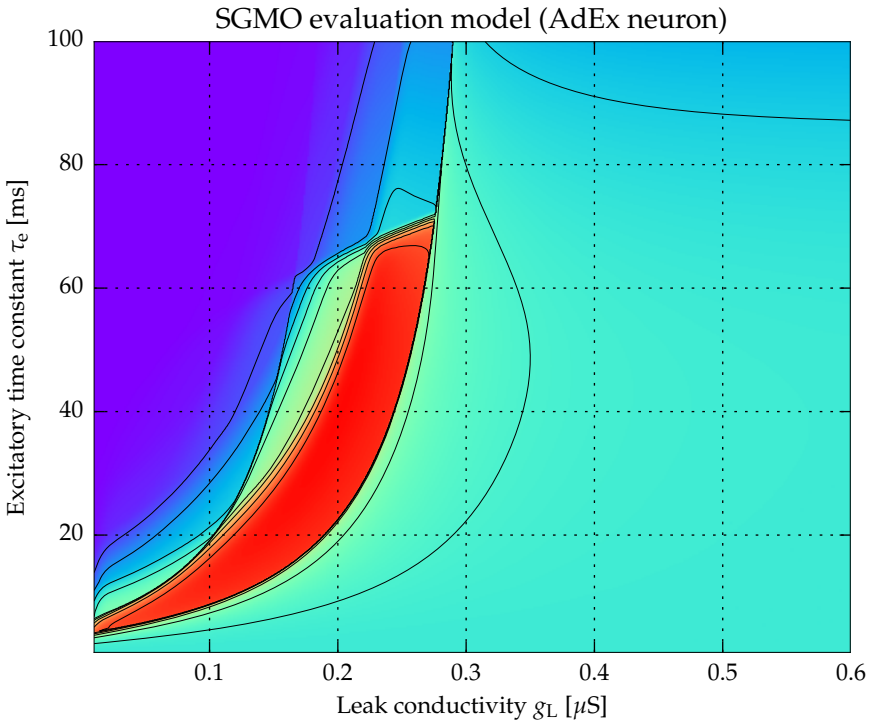
Integrator		*Time and samples*			*Error (RMSE)*								
		t [ms]	N	$\frac{t}{N}$ [µs]	u [mV]	(%)	g_e [nS]	(%)	g_i [nS]	(%)	w [nA]	(%)	Avg. %
Euler	$h = 1$ µs	2244.048	10011577	0.224	0.361	(0.36)	0.001	(0.00)	0.000	(0.00)	0.001	(0.33)	0.17
	$h = 10$ µs	182.803	1002278	0.182	0.930	(0.93)	0.012	(0.01)	0.004	(0.01)	0.001	(0.72)	0.42
	$h = 100$ µs	15.416	101255	0.152	3.530	(3.53)	0.122	(0.11)	0.039	(0.07)	0.011	(6.60)	2.58
	$h = 1$ ms	1.651	11190	0.148	8.358	(8.36)	1.225	(1.07)	0.382	(0.64)	0.017	(10.66)	5.18
Midpoint	$h = 1$ µs	2994.206	10011577	0.299	0.306	(0.31)	0.000	(0.00)	0.000	(0.00)	0.000	(0.30)	0.15
	$h = 10$ µs	264.261	1002278	0.264	0.887	(0.89)	0.000	(0.00)	0.000	(0.00)	0.001	(0.70)	0.40
	$h = 100$ µs	23.706	101249	0.234	3.554	(3.55)	0.001	(0.00)	0.000	(0.00)	0.038	(23.67)	6.80
	$h = 1$ ms	2.579	11161	0.231	42.230	(42.23)	0.088	(0.08)	0.027	(0.05)	2.424	(1514.39)	389.19
Runge-Kutta	$h = 1$ µs	5045.481	10011577	0.504	0.294	(0.29)	0.000	(0.00)	0.000	(0.00)	0.000	(0.29)	0.15
	$h = 10$ µs	460.079	1002278	0.459	0.856	(0.86)	0.000	(0.00)	0.000	(0.00)	0.001	(0.70)	0.39
	$h = 100$ µs	47.700	101251	0.471	3.932	(3.93)	0.000	(0.00)	0.000	(0.00)	0.040	(24.81)	7.19
	$h = 1$ ms	4.560	11149	0.409	48.258	(48.26)	0.000	(0.00)	0.000	(0.00)	2.754	(1720.25)	442.13
Dormand-Prince	$e = 1$ µ	2989.163	1933991	1.546	0.294	(0.29)	0.000	(0.00)	0.000	(0.00)	0.000	(0.30)	0.15
	$e = 10$ µ	735.503	699604	1.051	0.285	(0.29)	0.000	(0.00)	0.000	(0.00)	0.000	(0.27)	0.14
	$e = 100$ µ	74.478	74522	0.999	0.323	(0.32)	0.000	(0.00)	0.000	(0.00)	0.001	(0.32)	0.16
	$e = 1$ m	10.839	10792	1.004	0.437	(0.44)	0.000	(0.00)	0.000	(0.00)	0.001	(0.36)	0.20
	$e = 10$ m	4.739	4504	1.052	2.164	(2.16)	0.003	(0.00)	0.001	(0.00)	0.006	(4.04)	1.55
	$e = 100$ m	4.205	3964	1.061	3.610	(3.61)	0.028	(0.02)	0.008	(0.01)	0.020	(12.18)	3.96

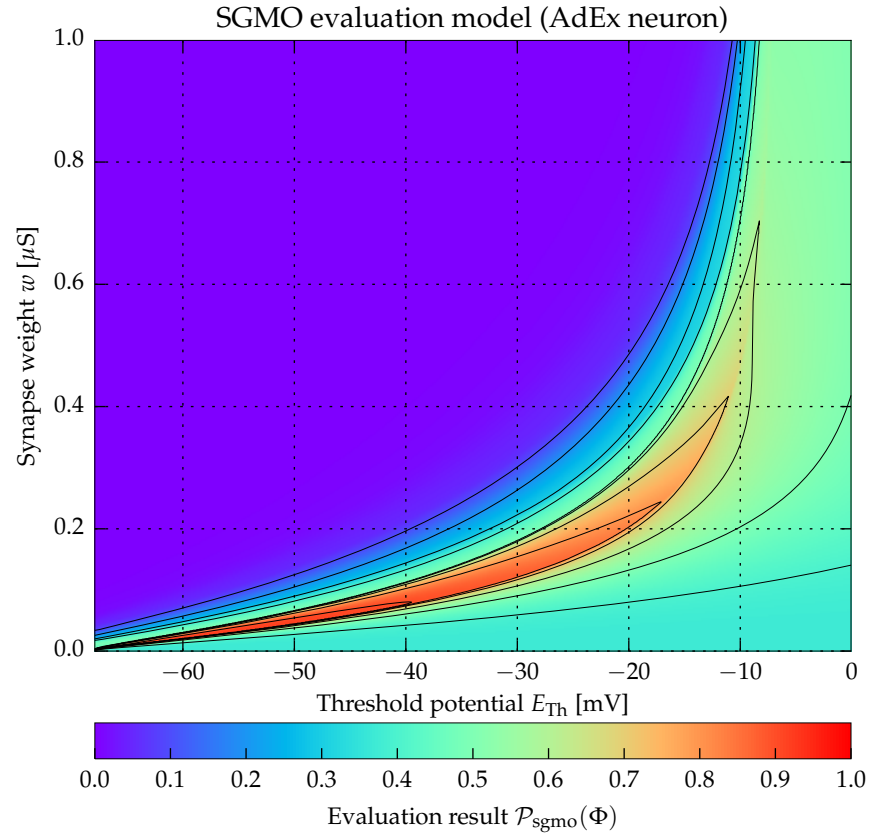
C

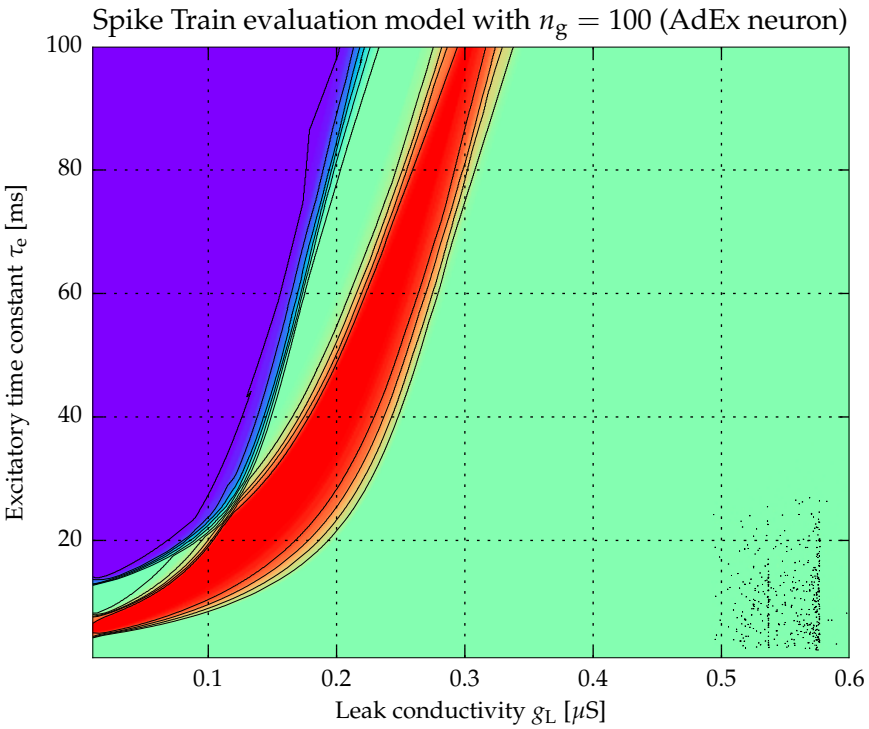
SINGLE NEURON EVALUATION COMPARISON

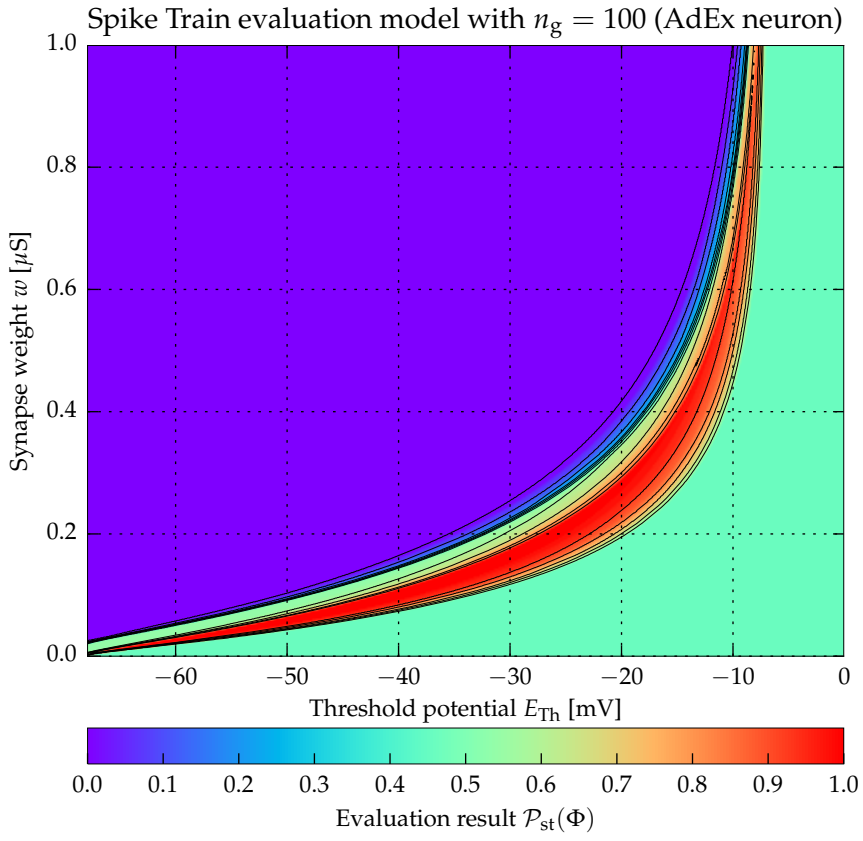


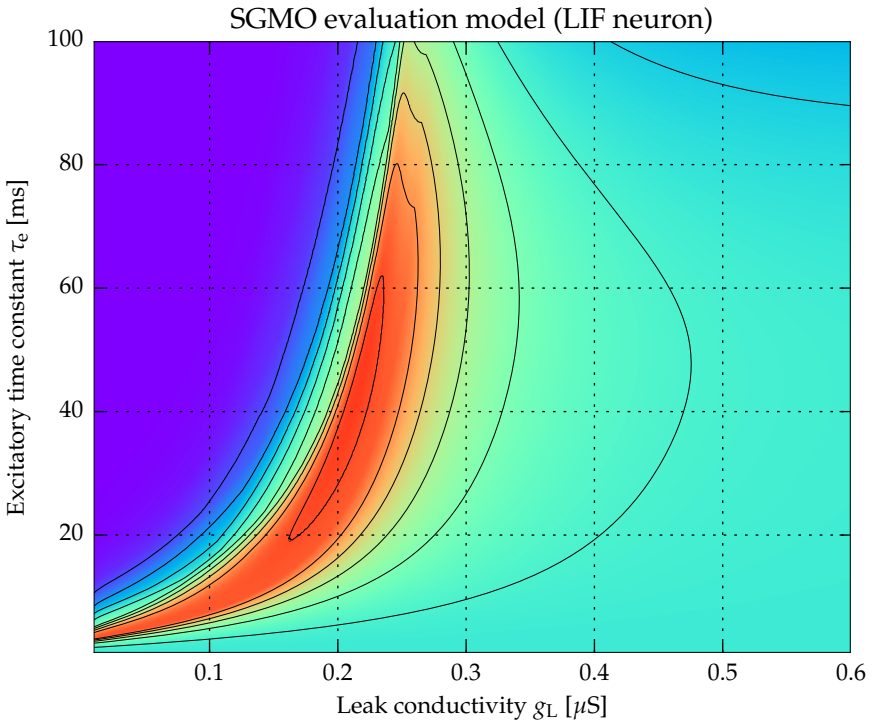


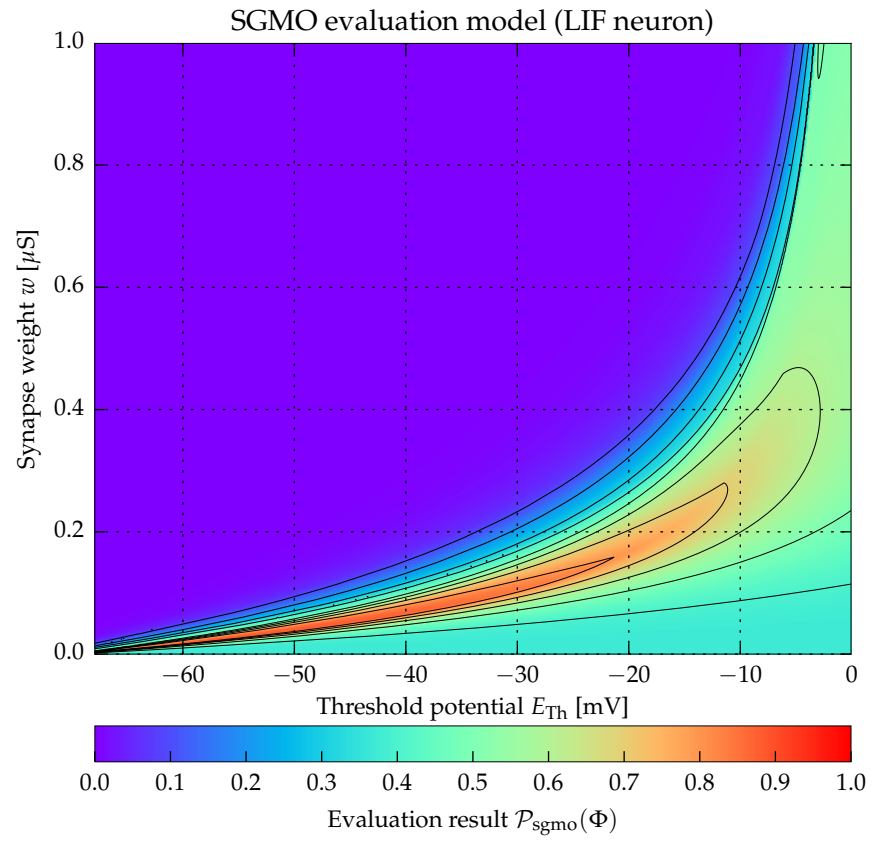

SGMO evaluation model (AdEx neuron)
Excitatory time constant τ_e [ms]
100
80
60
40
20
0.1
0.2
0.3
0.4
0.5
0.6
Leak conductivity g_L [μS]

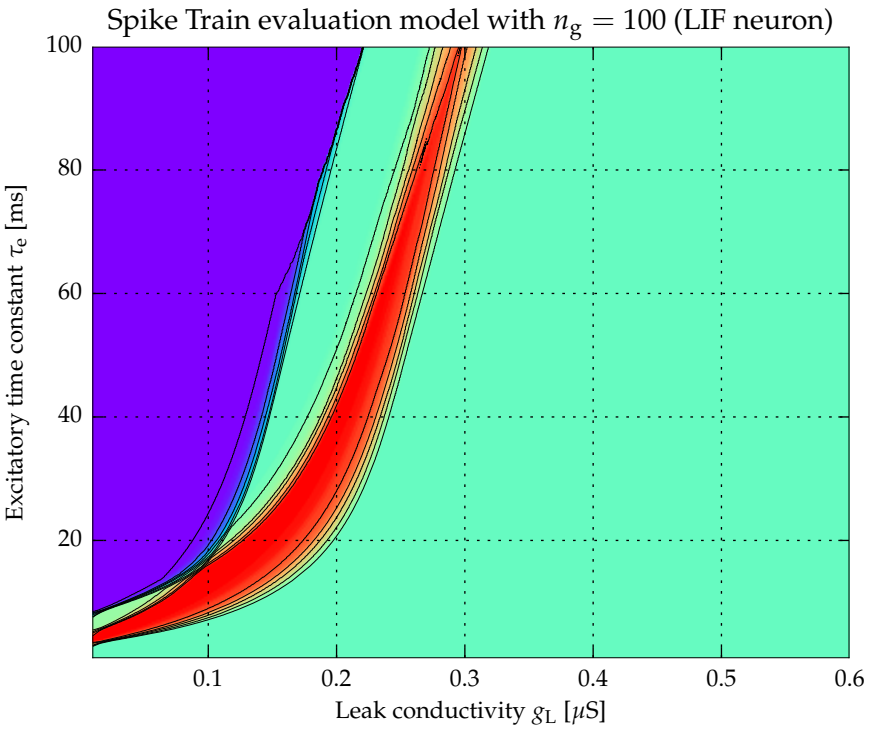

SGMO evaluation model (AdEx neuron)
Synapse weight w [μS]
1.0
0.8
0.6
0.4
0.2
0.0
−60
−50
−40
−30
−20
−10
0
Threshold potential E_{Th} [mV]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Evaluation result $\mathcal{P}_{sgmo}(\Phi)$

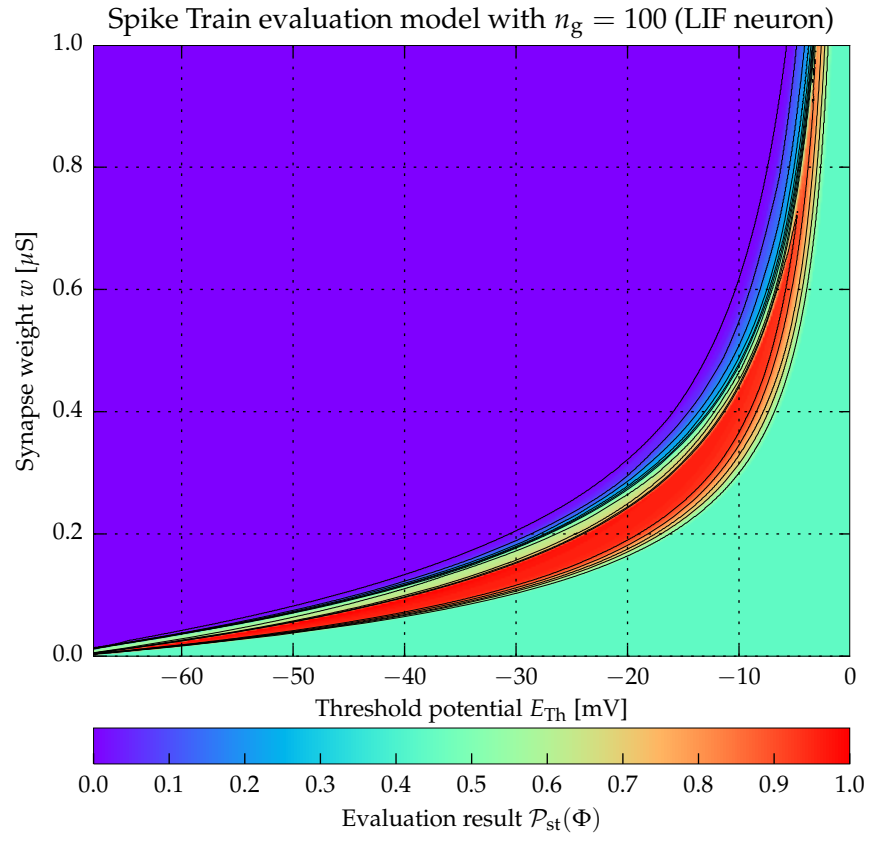

Spike Train evaluation model with $n_g = 100$ (AdEx neuron)
Excitatory time constant τ_e [ms]
100
80
60
40
20
0.1
0.2
0.3
0.4
0.5
0.6
Leak conductivity g_L [μS]

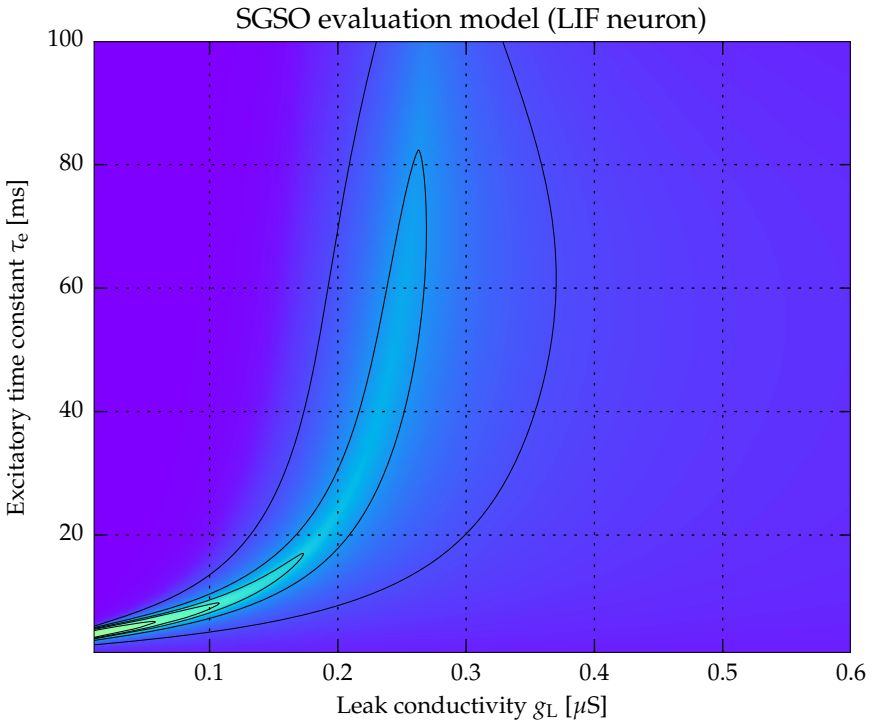

Spike Train evaluation model with $n_g = 100$ (AdEx neuron)
Synapse weight w [μS]
1.0
0.8
0.6
0.4
0.2
0.0
−60
−50
−40
−30
−20
−10
0
Threshold potential E_{Th} [mV]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Evaluation result $\mathcal{P}_{st}(\Phi)$

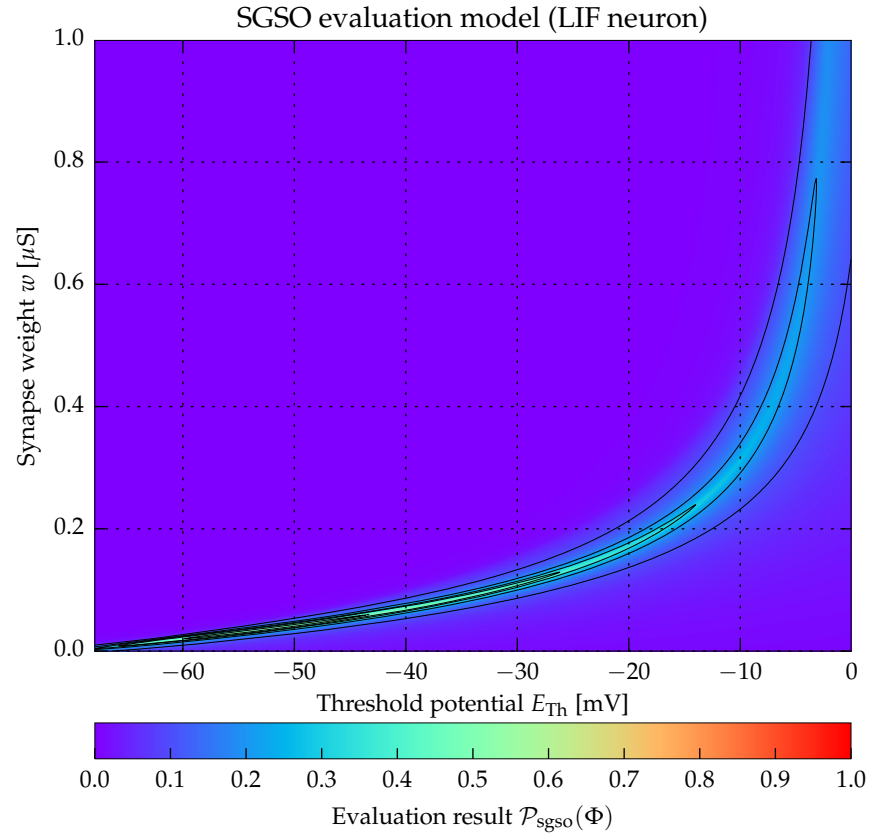

SGMO evaluation model (LIF neuron)
Excitatory time constant τ_e [ms]
100
80
60
40
20
0.1
0.2
0.3
0.4
0.5
0.6
Leak conductivity g_L [μS]


SGMO evaluation model (LIF neuron)
Synapse weight w [μS]
1.0
0.8
0.6
0.4
0.2
0.0
−60
−50
−40
−30
−20
−10
0
Threshold potential E_{Th} [mV]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Evaluation result $\mathcal{P}_{sgmo}(\Phi)$


Spike Train evaluation model with $n_g = 100$ (LIF neuron)
Excitatory time constant τ_e [ms]
100
80
60
40
20
0.1
0.2
0.3
0.4
0.5
0.6
Leak conductivity g_L [μS]


Spike Train evaluation model with $n_g = 100$ (LIF neuron)
Synapse weight w [μS]
1.0
0.8
0.6
0.4
0.2
0.0
−60
−50
−40
−30
−20
−10
0
Threshold potential E_{Th} [mV]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Evaluation result $\mathcal{P}_{st}(\Phi)$


SGSO evaluation model (LIF neuron)
Excitatory time constant τ_e [ms]
100
80
60
40
20
0.1
0.2
0.3
0.4
0.5
0.6
Leak conductivity g_L [μS]


SGSO evaluation model (LIF neuron)
Synapse weight w [μS]
1.0
0.8
0.6
0.4
0.2
0.0
−60
−50
−40
−30
−20
−10
0
Threshold potential E_{Th} [mV]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Evaluation result $\mathcal{P}_{sgso}(\Phi)$

ACRONYMS

AdEx	Adaptive exponential integrate-and-fire neuron model. Neuron model implemented on the NM-PM1 system. 21–23, 26, 38, 44, 50, 63, 64, 67–76, 80, 83, 90, 91, 93, 95, 96, 98, 126
AdExpSim	Adaptive exponential neuron simulator framework. Collection of libraries and applications for single neuron evaluation. 90, 92, 93, 99, 119, 121, 123
API	Application programming interface. Specification of a software interface (e. g. a collection of classes, data types and functions) which allows programmers to incorporate third-party systems into their applications. 27, 104
BiNAM	Binary neural associative memory, also known as Willshaw associative memory. 1, 4, 5, 28, 30–42, 44, 47–49, 52, 53, 56, 57, 61, 63–67, 77–79, 84, 94, 95, 101–103, 105, 107, 113, 117, 119, 120
BrainScaleS	Brain-inspired multiscale computation in neuromorphic hybrid systems. European research project with the goal (amongst others) to advance the wafer-scale HICANN system. A predecessor of the HBP. 25
CLI	Command line interface. 90, 92
CPU	Central processing unit. 96, 105, 126
DoF	Degree of freedom. 68, 69
EIF	Exponential integrate-and-fire neuron model. 20, 21
ESS	Executable system specification. A software emulator of the NM-PM1 system. 26, 27, 104, 106, 114–117
FACETS	Fast analogue computing with emergent transient states. European research project in aimed at the development of the Spikey and HICANN neuromorphic hardware systems. 25, 26
GPU	Graphics processing unit. Massively parallel processor system primarily designed for 3D graphics processing. In the recent years also usable for general purpose computing. 121
GUI	Graphical user interface. 90, 92
HBP	Human brain project. A European research project working towards an understanding of the human brain by constructing brain atlases and performing large-scale simulations of brain-like neural circuitry. 1–3, 7, 17, 19, 21, 24, 25, 27, 104, 106, 120, 121

HDF5 Hierarchical data format version 5. Storage format for the management of large and complex data collections. 104, 106

HH Hodgkin-Huxley neuron model. A detailed biophysical neuron model which sufficiently describes a variety of behavioural patterns of biological neurons. 11, 14, 16, 17, 21, 121

HICANN High input count analogue neural network. The actual analogue neural network chip on the NM-PM1 wafer. 2, 25, 26, 106, 120

IfCondExp Integrate-and-fire with conductance based exponential decay neuron model. Equivalent to the LIF model in conjunction with conductance based synapses with exponential decay. 19

ISI Interspike interval, denoted as Δt. 44

JSON JavaScript object notation, a text based format for the serialisation of data structures into a text based hierarchy of lists and dictionaries. 90, 104, 105

LIF Linear integrate-and-fire neuron model. One of the most simple neuron models, supported on all hardware platforms. 19–23, 25, 26, 38, 63, 64, 67, 68, 70, 72, 76, 80, 83, 88, 90, 91, 93, 95, 96, 98–101, 106, 107, 113, 114, 123, 124, 126

MAT Multi-timescale adaptive threshold neuron model. 21

MLP Multilayer perceptron. 10

NEST Neural simulation tool. Software simulator for spiking neural network models. 27, 104, 107–117

NM-MC1 Neuromorphic many-core system (version one). A digital spiking neural network simulator consisting of thousands of low-powered microprocessors, developed at the University of Manchester. 2, 24, 25, 27, 65, 68, 106–111, 113–117, 120

NM-PM1 Neuromorphic physical-model system (version one) developed at the Kirchhoff Institute for Physics at Heidelberg University. The NM-PM1 is a mixed-signal system which simulates the individual neurons with an analogue model circuit and uses digital routing infrastructure for inter-neuron spike propagation. 2, 21, 24–27, 65, 68, 92, 99, 104, 106, 114, 120

PyNAM Python neural associative memory framework. Tool for conducting network-level parameter sweeps on neuromorphic hardware. 103–107, 119, 120, 124

PyNN A Python package for simulator-independent specification of neuronal network models. 23, 27, 103, 104

PyNNLess Yet another python software abstraction layer on top of *PyNN*, developed for this thesis. Allows the execution of the same network descriptors on all platforms. 103, 104, 106, 119, 124

QIF Quadratic integrate-and-fire neuron model. 20

RMSE Root mean square error. The RMSE is defined as:

$$E = \sqrt{\frac{1}{N} \cdot \sum_{k=1}^{N} \|\vec{t}_k - \vec{x}_k\|^2},$$

where N is the number of samples, $\vec{t}_k$ are the reference samples and $\vec{x}_k$ the measured values. 75

SGMO Single group, multiple output spikes evaluation measure. 84, 85, 94–96, 98–102, 108, 109, 111, 113, 119, 121

SGSO Single group, single output spike evaluation measure. 79, 80, 84, 85, 94–96, 98–101, 108, 121

SQNR Signal-to-quantisation-noise ratio. 50

ST_{100} Spike train evaluation measure with $n_g = 100$ experiment groups. 95–101, 108, 109, 111, 112

ST_{10} Spike train evaluation measure with $n_g = 10$ experiment groups. 95–97

STDP Spike-timing dependent plasticity. Scheme for Hebbian training of synapse weights in spiking neural networks. 121

SYMBOLS

a Subthreshold adaptation conductance in the AdEx model in siemens. 21–23, 64, 67–69, 76, 95

α Dimensionality of the neuron state vector $\vec{v} \in \mathbb{R}^{\alpha}$. 73

$\mathbb{B}$ Base set of Boolean algebra, defined as $\mathbb{B} = \{0, 1\}$. 6, 9, 30–32, 54, 55, 59

B A binary matrix used as generalisation of X and Y in the data generation algorithms. The matrix contains either input or output vectors $\vec{x}_k, \vec{y}_k$ as rows. 54–56, 59

b Spike-triggered adaptation current in the AdEx model. 21–23, 64, 67–69, 95

c Number of ones in a memory input vector $\vec{x}$, defined as $c = \sum_i^m (\vec{x})_i$. 30–34, 36, 37, 46, 48, 52–54, 56, 64, 67, 78, 81, 99, 107

C_{m} Neuron membrane capacitance in farad. 12, 13, 17, 19–21, 23, 26, 64, 67–69, 72, 95, 98, 100, 106, 111, 114

$\mathfrak{D}$ Test dataset. Consists of N input and output vectors $\mathfrak{D} = \{\vec{x}_k, \vec{y}_k\}$. Input and output vectors are alternatively represented as the input and output matrices X and Y. 30, 31, 36, 52, 103, 105, 107

d Number of ones in a memory output vector $\vec{y}$, defined as $d = \sum_i^n (\vec{y})_i$. 30–34, 36, 46, 52–54, 56, 64, 67, 107

δ Latency, time until the entire output for an input has been produced. 51, 65, 107, 110, 113

Δ_{a} Adaptation induced voltage change, substitute of the state variable $I_{\mathrm{a}}(t)$. 69, 71, 73

Δ_{b} Spike induced adaptation, substitute of the parameter b. 69, 71

Δt Interspike interval (ISI), the equidistant delay between two spikes in a single spike burst in seconds. 22, 44–48, 64, 78, 81, 99

Δ_{Th} Spike slope factor in the AdEx model. 21–23, 64, 67–69, 72, 83, 95

Δu Perturbation potential used in the fractional spike count measure. 89

e Target approximation error for an adaptive step size integrator. 71, 74–76

E_{e} Excitatory reversal potential. Reversal potential of the excitatory conductance based synapses, usually chosen larger or equal to the threshold potential E_{Th}. 18, 19, 23, 64, 67–69, 95, 100, 106

E_{eq} Neuron membrane equilibrium potential: the potential the membrane is pulled towards given the ionic channel conductances. 12, 13

E_{i} Inhibitory reversal potential. Reversal potential of the inhibitory conductance based synapses, usually chosen smaller then the leak potential E_{L}. 18, 19, 23, 64, 67–69, 106

E_{L} Resting or leak potential. The membrane potential a neuron converges to over time. 11, 14, 19–21, 23, 51, 64, 67–69, 74, 83, 87, 95, 100, 106, 111

E_{reset} Reset potential. The membrane potential to neuron resets to following an output spike. 14, 19, 23, 64, 67–69, 71, 72, 79, 95, 100, 106

E_{spike} Spike potential, the maximum potential reached during a spike. 14, 19, 21, 22, 68

E_{Th} Threshold potential. The membrane potential that has to be passed for a spike to be generated. 14, 18–23, 64, 67–69, 71, 72, 80, 83, 91, 95, 96, 98, 100, 106, 109–112, 114

$E_{\mathrm{Th}}^{\mathrm{eff}}$ Effective threshold potential. Minimum membrane potential in non-linear integrate-and-fire models that has to be exceeded in order to surely trigger an output spike, given that there is no sudden rise in an external discharging/inhibitory current I_{syn}. 20, 79, 80, 82, 83, 87, 89, 98

$E_{\mathrm{Th}}^{\mathrm{exp}}$ Exponential threshold potential. Potential at which the inner term in the exponential of the AdEx model gets positive – at this point an avalanche effect will increase the membrane potential and lead to the production of a spike. 21, 23, 64, 67, 69, 72, 83, 95

f_{a} Subthreshold adaptation, substitute of the parameter a. 69

f_{e} Current excitatory channel frequency, substitute of the state variable $g_{\mathrm{e}}(t)$. 69, 71, 73

f_{i} Current inhibitory channel frequency, substitute of the state variable $g_{\mathrm{i}}(t)$. 69, 71, 73

f_{L} Membrane leak channel rate, substitute of the parameter g_{L}. 69

f_{w} Synaptic weight, substitute of the parameter w. 69

g_{e} Excitatory synapse conductance in siemens. 19, 23, 66, 69, 127–129

g_{i} Inhibitory synapse conductance in siemens. 19, 23, 69, 83, 127–129

g_{L} Conductance of the membrane leak channel in siemens. 19–21, 23, 26, 64, 67–69, 72, 83, 88, 91, 95–97, 100, 101, 106, 108, 111, 124

H Heaviside function $H(x)$, named after the mathematician and physicist Oliver Heaviside. 9, 10, 31

h Differential equation integrator step size. 71–76, 113

I Information (or entropy) stored in a memory in bits. 32–34, 37, 107–109, 111, 112, 114–117

$\mathcal{I}$ Place holder for an ion species with either positive or negative charge. 12, 13

i Total neuronal current which charges and discharges the cell membrane. The total neuronal current is composed of the synaptic current I_{syn} and the channel current I_{chan}. 17

I_{a} Adaptation current in the AdEx neuron model. 21, 23, 69, 72, 83, 90

I_{chan} Neuron model specific intrinsic component of the neuronal current i. 17, 18

I_{syn} Synaptic or external component of the neuronal current i. 16, 17, 19–21

I_{syn}^{k} Current induced by a single synapse k. 17, 18

I_{Th} Exponential threshold current used in the AdEx model. 21, 22, 72, 80, 83

$I_{\text{Th}}^{\max}$ Maximum the exponential threshold current I_{Th} is limited to for numerical integration. 72

K Population size. Number of neurons in the network output layer representing a single output component. 39, 46–48, 64, 78, 98, 99

k^{in} List of sample indices. Assigns a sample number to each input spike time in t^{in}. 47, 103

λ_{a} Adaptation channel decay rate, substitute of the parameter τ_{a}. 69

λ_{e} Excitatory channel decay rate, substitute of the parameter τ_{e}. 69

λ_{i} Inhibitory channel decay rate, substitute of the parameter τ_{i}. 69

M Binary matrix of size $m \times n$, storing the trained associations of the network. 30–32, 34–37, 39, 40, 53–55, 103

m Input dimensionality of the BiNAM. 30–36, 42, 46, 47, 52–55, 64, 67, 107

μ_t^{offs} Random offset chosen for an entire spike burst. This value is once sampled for a spike burst from a Gaussian distribution with standard deviation σ_t^{offs}. 45

N Number of samples trained in the BiNAM. 30, 33, 34, 36, 46, 47, 51–61, 64, 107, 141

n Output dimensionality of the BiNAM. 30, 32–36, 42, 46, 52–54, 64, 66, 67, 107

n_0 Maximum number of input spikes for which a neuron in the spiking BiNAM implementation should produce no output spikes. 48, 79, 80, 95, 99

n_1 Minimum number of input spikes for which a neuron in the spiking BiNAM implementation should produce s^{out} output spikes. 48, 79, 80, 95, 99

n_{E} Number of excitatory input bursts in an experiment group descriptor of the "spike train" evaluation measure. 77, 78

n_{I} Number of inhibitory input bursts in an experiment group descriptor of the "spike train" evaluation measure. 77, 78

n_{fn} Total number of false negative bits across the entire test dataset. 107, 110, 116

n_{fn}^{k} Number of false negative bits for the recall of the k-th sample. 33, 49

n_{fp} Total number of false positive bits across the entire test dataset. 107, 110–112

n_{fp}^{k} Number of false positive bits for the recall of the k-th sample. 33, 34, 49

n_g Total number of experiment groups in the "spike train" evaluation measure. 77–79, 94–96

n^{out} Actual number of output spike bursts. 84–90

n_i^{out} Actual number of receviced output spikes in the i-th experiment group. 78, 79

$\tilde{n}^{out}$ Expected number of output spike bursts in an experiment group descriptor of the "spike train" evaluation measure. 77, 78, 84, 85, 87, 94, 99

$\tilde{n}_i^{out}$ Expected number of output spikes in the i-th experiment group. 78, 79

o Fractional target spike count offset. Used to account for non-linear fractional values connecting the upper corners of the underlying step function. 84, 85

$\mathcal{P}$ Abstract optimality value returned by a single neuron optimisation measure such as the spike train measure, the SGSO measure or the SGMO measure.. 76, 96, 97, 100, 108, 109, 112

p_0 Probability of an input entity (either a single bit or a spike) being removed from the input data (false-negative). 36, 37, 44, 46, 47, 50, 64, 99

p_1 Probability of a false-positive input (either a single bit or a spike) being added to the input data. 36, 37, 44, 46, 47, 50, 64, 99

Φ Abstract spiking neuron parameter vector. 62–66, 76–79, 82, 84–86, 89, 94, 98, 103, 114

ϕ A single abstract spiking neuron parameter. 50, 86, 87

p^{out} Fractional component of the fractional spike count measure, it holds $p^{out} \in [0, 1) \subset \mathbb{R}$. 84, 86–90

$\mathcal{P}_{sgmo}$ Result of the "single group, multiple output spikes" evaluation method. 85, 94

$\mathcal{P}_{sgso}$ Result of the "single group, single output spike" evaluation method. 79, 81, 82, 94

$\mathcal{P}_{st}$ Result of the "spike train" evaluation method. $\mathcal{P}_{st}$ is defined as the ratio between the number of experiment groups for which the actual number of output spikes equals the expected number of output spike ands the total number of experiment groups. 77, 79, 101

$\mathcal{P}_{st}^i$ Partial result of the $\mathcal{P}_{st}$ evaluation measure for the i-th experiment group. 79

q^{out} Fractional output spike count, it holds $q^{out} \in \mathbb{R}^+$. 84–86, 88, 90

σ_ϕ Standard deviation of a single neuron parameter ϕ. 64

σ_t Jitter, standard deviation of the Gaussian distribution the spikes of a single spike burst are selected from. 45–47, 50, 64, 66, 75, 78, 80, 81, 99, 114–117

σ_t^{offs} Standard deviation of the random offset μ_t^{offs} for an entire burst. 45–47, 50, 64, 78, 80, 81, 99

σ_w Standard deviation of the Gaussian noise added to the synaptic weight w, special case of σ_ϕ. 78, 99, 114, 116

s^{in} Input burst size, the number of spikes in an input burst used to convey the information of a "one" being sent. 44–48, 64, 77, 98, 99

s^{out} Output burst size, the number of spikes expected from the network representing a "one". 44, 46, 48, 49, 64, 78, 98, 99

T Time window size, a group of input and output spikes belonging together has to fit within this time window. 43, 45–48, 51, 64, 75, 77, 78, 80, 82, 84, 85, 99, 105, 114, 116, 117

τ_{a} Adaptation current time constant, controls the exponential decay of the adaptation current I_{a} (in seconds). 21, 23, 64, 67–69, 95

τ_{e} Excitatory synapse time constant, controls the exponential decay of excitatory channel conductance g_{e}. 18, 19, 23, 64, 67–69, 95–97, 100, 101, 106, 108, 111, 124

τ_{i} Inhibitory synapse time constant, controls the exponential decay of inhibitory channel conductance g_{i}. 18, 19, 23, 64, 67–69, 106

τ_{m} Membrane potential time constant, controls the exponential decay of the membrane potential if no external input current is present. It holds $\tau_{\text{m}} = C_{\text{m}}/g_{\text{L}}$. Measured in seconds. 23

τ_{ref} Refractory period. Period of time for which a neuron produces no further spikes. In the simple neuron models the membrane potential is clamped to E_{reset} during the refractory period. 19, 23, 26, 64, 67–69, 71, 89, 91, 95, 100, 106

Θ Vectorial threshold function. $\Theta_\theta(\vec{z})$ returns a copy of $\vec{z}$ with all components greater of equal to θ set to one and all other values set to zero. 31, 34

θ Threshold value. Used in conjunction with McCulloch-Pitts neurons to specify the minimum neuronal excitation which causes a "one" at the output. 31, 32, 34–37

t^{in} Single input spike train in the case of single neuron simulation, or a list of input spike trains for each memory input component. 47, 66, 70, 71, 77, 78, 86, 103, 105

t_i^{in} Input spike train for the i-th memory input component, or the i-th spike in a single neuron evaluation output spike train. 42

t^{out} Single output spike train in the case of single neuron simulation, or list of output spike trains for each memory output component. 71, 78, 89, 103

t_j^{out} Out spike train for the j-th memory input component, or the j-th spike in a single neuron evaluation output spike train. 42, 78

u Neuron membrane potential. 11–14, 16–21, 23, 51, 65, 69, 71–75, 79, 80, 83, 87–89, 127–129

$\dot{u}$ Time differential of the neuron membrane potential. 13, 17, 19–21, 71, 72, 87

$\ddot{u}$ Second order time differential of the neuron membrane potential. 87

u_{max} Maximum neuron membrane potential encountered during an experiment. 79, 80

$\vec{v}$ Neuron state vector, $\vec{v} \in \mathbb{R}^\alpha$. For the AdEx model with conductance based excitatory and inhibitory synapses, the state vector is four dimensional, consisting of the membrane potential, the adaptation current and the excitatory and inhibitory channel conductances. 70, 71, 73, 74, 89, 123

$\vec{v}_0$ Initial neuron state at the beginning of the simulation. 73, 82, 85

w Synapse weight. For conductance based synapses the weight is measured in siemens, for current based synapses in ampere. 17, 18, 64, 66–69, 78, 95, 96, 99, 100, 106, 108–112, 114

w_E Weight factor to be applied to the excitatory synapse in the "spike train" evaluation measure. 78

w_I Weight factor to be applied to the inhibitory synapse in the "spike train" evaluation measure. 78

w^in Synaptic weight annotations for each input spike in single neuron simulation. 66, 70, 71, 77, 78

X Input data matrix, consists of N input vectors organised in rows. 52, 54, 55

$\vec{x}$ m-dimensional binary input vector of the BiNAM. 28–37, 40–43, 45, 47, 48, 50, 52–54, 57, 62, 107, 141

Y Output data matrix, consists of N output vectors organised in rows. 52, 54, 55

$\vec{y}$ n-dimensional binary output vector of the BiNAM. 29–36, 40–44, 46, 49, 50, 52, 53, 57, 62, 65, 107, 141

$\hat{y}_k$ Actual output of the BiNAM for a previously trained sample $\vec{x}_k$. 49

$\tilde{y}$ Intermediate result of the matrix-vector multiplication $M \cdot \vec{x}$ in the BiNAM recall rule. 31, 32

BIBLIOGRAPHY

[Bel57] Richard Ernest Bellman. *Dynamic Programming*. 6. print. 1957.

[BF87] Jon Bentley and Robert Floyd. "Programming pearls: a sample of brilliance". In: *Communications of the ACM* 30.9 (1987), pp. 754–757.

[BG05] Romain Brette and Wulfram Gerstner. "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity". In: *Journal of Neurophysiology* 94.5 (2005), pp. 3637–3642.

[BH97] Boris Barbour and Michael Hausser. "Intersynaptic diffusion of neurotransmitter". In: *Trends in neurosciences* 20.9 (1997), pp. 377–384.

[Bra14] *The JavaScript Object Notation (JSON) Data Interchange Format*. Proposed Standard RFC7159. Internet Engineering Task Force, 2014.

[Brü+11] Daniel Brüderle et al. "A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems". In: *Biological cybernetics* 104.4-5 (2011), pp. 263–296.

[BS13] Valentino Braitenberg and Almut Schüz. *Cortex: statistics and geometry of neuronal connectivity*. Springer Science & Business Media, 2013.

[Caj04] Santiago Ramón Y. Cajal. *Textura del Sistema Nervioso del Hombre y de los Vertebrados*. Vol. 2. Nicolas Moya, Madrid, 1904.

[CG90] Barry W. Connors and Michael J. Gutnick. "Intrinsic firing patterns of diverse neocortical neurons". In: *Trends in neurosciences* 13.3 (1990), pp. 99–104.

[Cor+96] Robert M. Corless et al. "On the Lambert W function". In: *Advances in Computational mathematics* 5.1 (1996), pp. 329–359.

[CP96] B. Jack Copeland and Diane Proudfoot. "On Alan Turing's anticipation of connectionism". English. In: *Synthese* 108.3 (1996), pp. 361–377.

[Dav+08] Andrew P. Davison et al. "PyNN: a common interface for neuronal network simulators". In: *Frontiers in neuroinformatics* 2 (2008).

[DP80] John R. Dormand and Peter J. Prince. "A family of embedded Runge-Kutta formulae". In: *Journal of computational and applied mathematics* 6.1 (1980), pp. 19–26.

[Eic15] Emanuel Eichhammer. *Qt Plotting Widget QCustomPlot*. 2015. URL: `http://www.qcustomplot.com/`.

[Fur+13] Steve B. Furber et al. "Overview of the SpiNNaker system architecture". In: *Computers, IEEE Transactions on* 62.12 (2013), pp. 2454–2467.

[GB09] Wulfram Gerstner and Romain Brette. "Adaptive exponential integrate-and-fire model". In: *Scholarpedia* 4.6 (2009), p. 8427.

[GD07] Marc-Oliver Gewaltig and Markus Diesmann. "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4 (2007), p. 1430.

[GK02] Wulfram Gerstner and Werner M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[Gli06] Mitch Glickstein. "Golgi and Cajal: The neuron doctrine and the 100th anniversary of the 1906 Nobel Prize". In: *Current Biology* 16.5 (2006), pp. 147–151.

[Hay11] S. O. Haykin. *Neural Networks and Learning Machines*. Pearson Education, 2011.

[HDF15] HDF Group. *Hierarchical Data Format, version 5*. 1997-2015. URL: `http://www.hdfgroup.org/HDF5/`.

[Heb05] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[HH52] Alan L. Hodgkin and Andrew F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), pp. 500–544.

[HM13] Jennifer Hasler and Bo Marr. "Finding a roadmap to achieve large neuromorphic hardware systems". In: *Frontiers in neuroscience* 7 (2013).

[HMW60] John C. Hay, F. C. Martin, and C. W. Wightman. "The Mark I Perceptron – Design and performance". In: *Proceedings of the institute of radio engineers*. Vol. 48. 3. 1960, pp. 398–399.

[Hop07] John J. Hopfield. "Hopfield network". In: *Scholarpedia* 2.5 (2007), p. 1977.

[Hop82] John J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[Hum15a] Human Brain Project. *Mission – The Human Brain Project*. 2015. URL: `https://www.humanbrainproject.eu/discover/the-project/strategic-objectives`.

[Hum15b] Human Brain Project. *Overview – The Human Brain Project*. 2015. URL: `https://www.humanbrainproject.eu/discover/the-project/overview`.

[Hum15c] Human Brain Project, SP9. *Neuromorphic Platform Specification – Public Version*. 2015. URL: `https://flagship.kip.uni-heidelberg.de/jss/FileExchange?fID=359&s=qqdXDg6HuX3&uID=65`.

[ISO14] ISO. *Information technology – Programming languages – C++*. Standard 14882:2014(E). Geneva, Switzerland: International Organization for Standardization, 2014.

[Izh04] Eugene M. Izhikevich. "Which model to use for cortical spiking neurons?" In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070.

[Izh07] Eugene M. Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007.

[JL07] Christopher Johansson and Anders Lansner. "Towards cortex sized artificial neural systems". In: *Neural Networks* 20.1 (2007), pp. 48–61.

[Kan+12] E. Kandel et al. *Principles of Neural Science, Fifth Edition*. McGraw-Hill Education, 2012.

[Kar72] Richard M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[Kno+14] Andreas Knoblauch et al. "Structural Synaptic Plasticity Has High Memory Capacity and Can Explain Graded Amnesia, Catastrophic Forgetting, and the Spacing Effect". In: (2014).

[Kno03] Andreas Knoblauch. *Synchronization and pattern separation in spiking associative memories and visual cortical areas.* Universität Ulm, Fakultät für Informatik, 2003.

[Knu98] Donald Ervin Knuth. *Sorting and searching*. 2. ed., 1. print. 1998, pp. 492–512.

[Koh12] Teuvo Kohonen. *Content-addressable memories*. Vol. 1. Springer Science & Business Media, 2012.

[KTS09] Ryota Kobayashi, Yasuhiro Tsubo, and Shigeru Shinomoto. "Made-to-order spiking neuron model equipped with a multi-timescale adaptive threshold". In: *Frontiers in computational neuroscience* 3 (2009).

[LD09] Bhagwandas L. Lathi and Zhi Ding. *Modern Digital and Analog Communication Systems*. Oxford University Press, 2009.

[Lis97] John E. Lisman. "Bursts as a unit of neural information: making unreliable synapses reliable". In: *Trends in neurosciences* 20.1 (1997), pp. 38–43.

[Maa97] Wolfgang Maass. "Networks of spiking neurons: the third generation of neural network models". In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[Mel10] Arnaldo Carvalho de Melo. "The new linux `perf` tools". In: *Slides from Linux Kongress*. 2010.

[Mie98] Kaisa Miettinen. *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media, 1998.

[Min12] Paul Mineiro. *Approximate and vectorized versions of functions commonly used in machine learning*. 2012. URL: `https://code.google.com/p/fastapprox/`.

[Mor+07] Abigail Morrison et al. "Exact subthreshold integration with continuous spike times in discrete-time neural network simulations". In: *Neural computation* 19.1 (2007), pp. 47–79.

[MP43] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[MP87] Marvin L. Minsky and Seymour A. Papert. *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press Boston, MA: 1987.

[MS95] Zachary F. Mainen and Terrence J. Sejnowski. "Reliability of spike timing in neocortical neurons". In: *Science* 268.5216 (1995), pp. 1503–1506.

[MT11] Eve Marder and Adam L. Taylor. "Multiple models to capture the variability in biological neurons and networks". In: *Nature neuroscience* 14.2 (2011), pp. 133–138.

[Mül15] Sylvia Müller. *Programmierung neuronaler Assoziativspeicher auf Basis pulsender Neuronen*. Bachelor Thesis. Cognitronics and Sensor Systems Group, Bielefeld University, 2015.

[NM65] John A. Nelder and Roger Mead. "A simplex method for function minimization". In: *The computer journal* 7.4 (1965), pp. 308–313.

[Pai+13] Eustace Painkras et al. "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation". In: *Solid-State Circuits, IEEE Journal of* 48.8 (2013), pp. 1943–1953.

[Pal13] Günther Palm. "Neural associative memories and sparse coding". In: *Neural Networks* 37 (2013), pp. 165–171.

[Pal80] Günther Palm. "On associative memory". In: *Biological cybernetics* 36.1 (1980), pp. 19–31.

[PBM04] Astrid A. Prinz, Dirk Bucher, and Eve Marder. "Similar network activity from disparate circuit parameters". In: *Nature neuroscience* 7.12 (2004), pp. 1345–1352.

[Pea01] JMS Pearce. "Emil Heinrich Du Bois-Reymond (1818–96)". In: *Journal of Neurology, Neurosurgery & Psychiatry* 71.5 (2001), pp. 620–620.

[Pet+14] Mihai A. Petrovici et al. "Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms". In: (2014).

[Pfe+13] Thomas Pfeil et al. "Six networks on a universal neuromorphic computing substrate". In: *Frontiers in Neuroscience* 7 (2013), p. 11.

[Pic97] Marco Piccolino. "Luigi Galvani and animal electricity: two centuries after the foundation of electrophysiology". In: *Trends in neurosciences* 20.10 (1997), pp. 443–448.

[Pre+07a] William H. Press et al. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. Chap. 17.

[Pre+07b] William H. Press et al. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. Chap. 10.

[Pri07] A. A. Prinz. "Neuronal parameter optimization". In: *Scholarpedia* 2.1 (2007), p. 1903.

[Qt 15] Qt Company. *Qt project homepage*. 2015. URL: `http://www.qt.io/`.

[Ras+10] Alexander D. Rast et al. "The leaky integrate-and-fire neuron: A platform for synaptic model exploration on the SpiNNaker chip". In: *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE. 2010, pp. 1–8.

[Red11] M. Reddy. *API Design for C++*. Elsevier Science, 2011.

[RS91] Ulrich Rückert and Hartmut Surmann. "Tolerance of a binary associative memory towards stuck-at-faults". In: *Artificial Neural Networks* 2 (1991).

[Sch+10] Johannes Schemmel et al. "A wafer-scale neuromorphic hardware system for large-scale neural modeling". In: *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE. 2010, pp. 1947–1950.

[Sch15] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117.

[Sch83] Stephen M. Schuetze. "The discovery of the action potential". In: *Trends in Neurosciences* 6 (1983), pp. 164–168.

[Sch99] Nicol N. Schraudolph. "A fast, compact approximation of the exponential function". In: *Neural Computation* 11.4 (1999), pp. 853–862.

[SG10] J. Sjöström and W. Gerstner. "Spike-timing dependent plasticity". In: *Scholarpedia* 5.2 (2010), p. 1362.

[Sha48] Claude Elwood Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27 (1948), pp. 379–423.

[SK93] William R. Softky and Christof Koch. "The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs". In: *The Journal of Neuroscience* 13.1 (1993), pp. 334–350.

[SN94] Michael N. Shadlen and William T. Newsome. "Noise, neural codes and cortical organization". In: *Current opinion in neurobiology* 4.4 (1994), pp. 569–579.

[Ste61] Karl Steinbuch. "Die Lernmatrix". In: *Biological Cybernetics* 1.1 (1961), pp. 36–45.

[Sto+93] J. Stoer et al. *Introduction to Numerical Analysis*. Texts in Applied Mathematics. Springer New York, 1993.

[Str13] Bjarne Stroustrup. *The C++ Programming Language*. 4th. Pearson Education, 2013, p. 700.

[War16] Howard C. Warren. "Mental association from Plato to Hume". In: *Psychological Review* 23.3 (1916), p. 208.

[WBL69] David J. Willshaw, O. Peter Buneman, and Hugh Christopher Longuet-Higgins. "Non-holographic associative memory." In: *Nature* (1969).

[WH60] Bernard Widrow and Marcian E. Hoff. "Adaptive switching circuits". In: *WESCON Convention Record Part IV*. 1960, pp. 96–104.

ERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbständig verfasst und gelieferte Datensätze, Zeichnungen, Skizzen und graphische Darstellungen selbständig erstellt habe. Ich habe keine anderen Quellen als die angegebenen benutzt und habe die Stellen der Arbeit, die anderen Werken entnommen sind – einschließlich verwendeter Tabellen und Abbildungen – in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Bielefeld, den 21. Dezember 2015 _____

(Unterschrift)