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
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 Paasen, Jan Philip Göpfert, Adriana Dreyer, Christoph Jenzen and Daniel Stöckel.
# Contents

List of Figures xiii  
List of Tables xv  
List of Algorithms xv  

1 Introduction 1  

1.1 Motivation and goals 1  
1.1.1 Neuromorphic hardware systems in the Human Brain Project 1  
1.1.2 Willshaw associative memory as a spiking neural network 2  
1.1.3 Associative memories as hardware benchmark 3  
1.1.4 Goals of this thesis 4  
1.2 Structure 4  
1.3 Notational conventions 5  

2 Background and Related Work 7  

2.1 History of artificial neural network models 7  
2.1.1 First generation: binary McCulloch-Pitts cells 8  
2.1.2 Second generation: firing-rate coded neural networks 9  
2.1.3 Third generation: spiking neural networks 10  
2.2 Biophysical neuron model 11  
2.2.1 Passive electrophysiological properties of the neuron membrane 11  
2.2.2 Action potentials 14  
2.2.3 Chemical synapses 15  
2.3 Simplified neuron and synapse models 16  
2.3.1 Neuron model base equation 17  
2.3.2 Synapse models 17  
2.3.3 Excitatory and inhibitory synapses 18  
2.3.4 Linear integrate-and-fire neuron model 19  
2.3.5 Non-linear integrate-and-fire models 19  
2.3.6 Two-dimensional Hodgkin-Huxley approximations: the AdEx model 21  
2.4 Neuromorphic hardware 24  
2.4.1 NM-MC1: The many-core system 24  
2.4.2 NM-PM1: The physical model 25  
2.4.3 Spikey 26  
2.4.4 Software stack 27
2.5 The Willshaw associative memory model (BiNAM) 28
   2.5.1 Artificial associative memory models 29
   2.5.2 Formal description of the Willshaw model 30
   2.5.3 Choice of the threshold $\theta$ 32
   2.5.4 Storage capacity and sparsity 32
   2.5.5 Neural network implementation 34
   2.5.6 Impact of noise 36

2.6 Summary and outlook 37

3 Spiking Associative Memory Architecture and Testing 39
   3.1 Neural network topology and data encoding 39
      3.1.1 Input-/output spike sequences 40
      3.1.2 Data encoding and input noise parametrisation 43
      3.1.3 Neuron populations 45
      3.1.4 Required neuron behaviour 47
   3.2 Memory evaluation measures 48
      3.2.1 Storage capacity 48
      3.2.2 Robustness in case of noise 49
      3.2.3 Latency and throughput 50
      3.2.4 Energy 51
   3.3 Data generation 52
      3.3.1 Dataset parametrisation 52
      3.3.2 Expected behaviour in reaction to uncorrelated random data 52
      3.3.3 Random data generation algorithm 54
      3.3.4 Balanced data 55
      3.3.5 Balanced data generation algorithm 58
   3.4 Conclusion 61

4 Neuron Parameter Evaluation and Optimisation 63
   4.1 Design space exploration 63
      4.1.1 On the terms “design space” and “exploration” 63
      4.1.2 Full network evaluation 65
      4.1.3 Single neuron evaluation 65
      4.1.4 Parameter constraints and intra-dependencies 67
   4.2 Single neuron simulation 70
      4.2.1 Neuron simulation loop 70
      4.2.2 Numerical integration of the AdEx model 72
      4.2.3 Differential equation integrators 73
      4.2.4 Integrator benchmark 75
   4.3 Approach 1: spike train 76
      4.3.1 Concept 77
      4.3.2 Descriptor and input spike train generation 77
      4.3.3 Evaluation 78
   4.4 Approach 2: single group, single output spike 79
      4.4.1 Concept 79
      4.4.2 Deterministic input spike train generation 80
      4.4.3 Evaluation measure 81
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Photos of the Spikey neuromorphic hardware system</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Drawings by Santiago Ramón y Cajal</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Sketch of a McCulloch-Pitts artificial neuron</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Sketch of a schematised model neuron</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>McCulloch-Pitts artificial neuron as Boolean operators</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>Sketch of a classical artificial neuron</td>
<td>10</td>
</tr>
<tr>
<td>2.6</td>
<td>Membrane potential change caused by selectively permeable ion channel</td>
<td>11</td>
</tr>
<tr>
<td>2.7</td>
<td>Model equivalent circuit diagram of the neuron membrane for three ion channels</td>
<td>12</td>
</tr>
<tr>
<td>2.8</td>
<td>Membrane potential over time for varying membrane conductances</td>
<td>13</td>
</tr>
<tr>
<td>2.9</td>
<td>Annotated sketch of an action potential</td>
<td>14</td>
</tr>
<tr>
<td>2.10</td>
<td>Chemical synapse schematic</td>
<td>15</td>
</tr>
<tr>
<td>2.11</td>
<td>Sketches of spiking neuron behavioural patterns</td>
<td>16</td>
</tr>
<tr>
<td>2.12</td>
<td>One dimensional integrate-and-fire model bifurcation patterns</td>
<td>20</td>
</tr>
<tr>
<td>2.13</td>
<td>Comparison between the LIF and AdEx neuron models</td>
<td>22</td>
</tr>
<tr>
<td>2.14</td>
<td>NM-PM1 wafer high level architecture overview</td>
<td>25</td>
</tr>
<tr>
<td>2.15</td>
<td>Neuromorphic hardware system software stack and data flow</td>
<td>27</td>
</tr>
<tr>
<td>2.16</td>
<td>Pattern completion in a Hopfield associative memory</td>
<td>28</td>
</tr>
<tr>
<td>2.17</td>
<td>BiNAM training example</td>
<td>30</td>
</tr>
<tr>
<td>2.18</td>
<td>BiNAM recall example</td>
<td>31</td>
</tr>
<tr>
<td>2.19</td>
<td>BiNAM information and false positive count over number of trained samples</td>
<td>33</td>
</tr>
<tr>
<td>2.20</td>
<td>BiNAM pattern completion experiment</td>
<td>35</td>
</tr>
<tr>
<td>2.21</td>
<td>Neural network implementation of the BiNAM</td>
<td>35</td>
</tr>
<tr>
<td>2.22</td>
<td>Information measure and error count over noise</td>
<td>36</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison between a single neuron BiNAM implementation and its population counterpart</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Spiking BiNAM implementation and spike train nomenclature</td>
<td>41</td>
</tr>
<tr>
<td>3.3</td>
<td>Comparison between a classical and spiking neural network</td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td>Example of input sample pipelining</td>
<td>43</td>
</tr>
<tr>
<td>3.5</td>
<td>Sketch of a burst with increasing interspike interval</td>
<td>44</td>
</tr>
<tr>
<td>3.6</td>
<td>Parametrisation of an example input spike burst with jitter</td>
<td>45</td>
</tr>
<tr>
<td>3.7</td>
<td>Sketch of the critical time window measure</td>
<td>51</td>
</tr>
<tr>
<td>3.8</td>
<td>Estimated number of false positives</td>
<td>54</td>
</tr>
<tr>
<td>3.9</td>
<td>Example bit vectors and their set representation</td>
<td>55</td>
</tr>
<tr>
<td>3.10</td>
<td>Comparison of the BiNAM matrix occupancy for different data generation methods</td>
<td>56</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.11</td>
<td>Example of correlation introduced by data balancing</td>
<td>56</td>
</tr>
<tr>
<td>3.12</td>
<td>Random versus balanced data generation</td>
<td>57</td>
</tr>
<tr>
<td>3.13</td>
<td>Balanced data generation algorithm example 1</td>
<td>58</td>
</tr>
<tr>
<td>3.14</td>
<td>Linear correlation coefficient with and without index selection bias</td>
<td>59</td>
</tr>
<tr>
<td>3.15</td>
<td>Balanced data generation algorithm example 2</td>
<td>60</td>
</tr>
<tr>
<td>3.16</td>
<td>Balanced data generation algorithm example 3</td>
<td>61</td>
</tr>
<tr>
<td>4.1</td>
<td>Membrane potential over time for a neuron with degenerate parameters causing low latency</td>
<td>65</td>
</tr>
<tr>
<td>4.2</td>
<td>Sketch of the input spike train fusion model</td>
<td>66</td>
</tr>
<tr>
<td>4.3</td>
<td>Example of a spike train evaluation</td>
<td>77</td>
</tr>
<tr>
<td>4.4</td>
<td>Conceptual overview of the single group, single output spike measure</td>
<td>80</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison of single neuron simulation spike train generation methods</td>
<td>81</td>
</tr>
<tr>
<td>4.6</td>
<td>Examples of heavy-tailed sigmoids and the threshold evaluation</td>
<td>82</td>
</tr>
<tr>
<td>4.7</td>
<td>Idealised sketch of the SGMO measure</td>
<td>85</td>
</tr>
<tr>
<td>4.8</td>
<td>Sketch of the fractional spike count decomposition</td>
<td>86</td>
</tr>
<tr>
<td>4.9</td>
<td>Unsuccessful fractional spike count calculation strategy</td>
<td>88</td>
</tr>
<tr>
<td>4.10</td>
<td>Minimal membrane potential perturbation measure examples</td>
<td>91</td>
</tr>
<tr>
<td>4.11</td>
<td>Architectural overview of the AdExpSim framework</td>
<td>92</td>
</tr>
<tr>
<td>4.12</td>
<td>Screenshots of the AdExpSimGui tool</td>
<td>93</td>
</tr>
<tr>
<td>4.13</td>
<td>Design space exploration results</td>
<td>97</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of the full network evaluation software stack</td>
<td>104</td>
</tr>
<tr>
<td>5.2</td>
<td>First NM-MC1 and NEST neuron parameter sweep</td>
<td>108</td>
</tr>
<tr>
<td>5.3</td>
<td>Second NM-MC1 and NEST neuron parameter sweep</td>
<td>109</td>
</tr>
<tr>
<td>5.4</td>
<td>NM-MC1 and NEST errors and latencies</td>
<td>110</td>
</tr>
<tr>
<td>5.5</td>
<td>Spikey parameter sweep overview</td>
<td>112</td>
</tr>
<tr>
<td>5.6</td>
<td>Spikey parameter sweep latencies</td>
<td>113</td>
</tr>
<tr>
<td>5.7</td>
<td>Results of the spike time noise parameter sweep</td>
<td>115</td>
</tr>
<tr>
<td>5.8</td>
<td>Results of the synaptic weight noise and time window sweeps</td>
<td>116</td>
</tr>
</tbody>
</table>
LIST OF TABLES

2.1 LIF and AdEx model parameters and state variables ... 23
3.1 Network and input/output parametrisation ............... 46
4.1 Overview of variables in the design space ............... 64
4.2 Nominal parameter constraints of the NM-PM1 platform 68
4.3 Overview of parameters and state variables in the AdEx
   model with reduced DoF ................................ 69
4.4 Neuron simulator benchmark results ..................... 75
4.5 Spike train evaluation method meta-parameter list ....... 78
4.6 Group descriptor example ................................ 78
4.7 Expected behaviour of the evaluation methods .......... 94
4.8 Initial neuron parameters ............................... 95
4.9 Neuron parameter space exploration experiment runtimes 96
4.10 Scenarios for neuron parameter optimisation ........... 99
4.11 Results of the neuron parameter optimisation experiment 100
4.12 Optimised LIF neuron parameters ..................... 100
B.1 Constant step size integrator Runge-Kutta coefficients .. 125
B.2 Adaptive step size integrator Runge-Kutta coefficients .. 125
B.3 Simulator profile with IEEE 754 exponential ............ 126
B.4 Simulator profile with approximated exponential ....... 126
B.5 Integrator performance for the AdExp model ............ 127
B.6 Integrator performance for the IfCondExp model ....... 128
B.7 Integrator performance for the AdExp model with approxi-
   mated exponential ...................................... 129

LIST OF ALGORITHMS

3.1 Input burst generation algorithm ....................... 47
3.2 Uncorrelated random data generation .................. 55
3.3 Uncorrelated, random and unique data generation ..... 59
4.1 Basic single neuron simulator loop .................... 71
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
**Introduction**

(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 $8 \times 7 \times 3$ 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).

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

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.

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 abnor-

mal 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.

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 $i, j, k, \ell$ 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.

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.
**Sets and tuples** Sets are usually typeset in fraktur (e.g., \( \mathcal{B} \)). 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 \( a \parallel b \) is then defined as:

\[
a \parallel b = (a_1, \ldots, a_i, b_1, \ldots, b_j)
\] (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 \( \delta(t) \):

\[
\int_{-\infty}^{\infty} f(t) \cdot \delta(t) \, dt = f(0).
\] (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

\[
\dot{u}(t) = g \cdot (u(t) - u_0),
\]

\[
u(t) \leftarrow u(t) + \Delta u \quad \text{if} \quad t = t_0,
\] (1.3) (1.4)

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

\[
\dot{u}(t) = g \cdot (u_0 - u(t)) + \delta(t - t_0) \cdot \Delta u.
\] (1.5)
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 liveliness – 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].
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

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
2.1 History of artificial neural network models

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)

\[
y = H(\vec{w}^\top \cdot \vec{x} - \theta)
\]

where \( H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \) (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].

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 \( \mathbf{w} \in \mathbb{R}^m \) 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 \( \mathbf{w} \) in such a way, that the perceptron outputs a desired answer \( y_k \) for a certain input \( \mathbf{x}_k \), allowing to solve linear classification and regression tasks [MP87].

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 \( \theta \) as an additional dimension (“bias”) into the input \( \mathbf{x} \) and the weights \( \mathbf{w} \) (Figure 2.5)

\[
y = f(\mathbf{w}^\top \cdot \mathbf{x}).
\]  

The weights \( \mathbf{w} \) 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 [IL07] and may in the future allow to simulate deep networks on specialised, energy efficient hardware [HM13; Sch15].
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

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 2.6: 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].
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 intracellular and extracellular ion concentration, an osmotic force causes $K^+$ to flow out of the cell. The total ionic current is proportional to the number of 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 $E_I$ for a membrane that is permeable for an ion species $I$ only. We refer to $E_I$ as the reversal potential for $I$: its ionic current reverses at this potential. For intra- and extracellular ion concentrations $[I]_{in}$ and $[I]_{out}$, $E_I$ is given according to the Nernst equation as

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

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 $g_I$ of the cell membrane for each ion species $I$, the equilibrium potential $E_{eq}$ can be calculated according to the Goldman–Hodgkin–Katz equation as

$$E_{eq} = \frac{\sum_I g_I \cdot E_I}{\sum_I g_I} = \frac{\sum_I g_I \cdot E_I}{g_{tot}}. \quad (2.4)$$

The behaviour modelled by Equation (2.4) corresponds to an electric circuit, consisting of parallel voltage sources with voltage $E_I$ and a series resistor with conductance $g_I$ 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 2.7: 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).
Figure 2.8: Membrane potential over time for varying membrane conductances, according to Equation (2.5). The reversal potentials are set to $E_{K^+} = -88$ mV, $E_{Na^+} = 61$ mV and $E_{Cl^-} = -60$ 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

$$-C_m \cdot \dot{u}(t) = i(t) = g_{tot} \cdot (u(t) - E_{eq}) = \sum_{I} g_I \cdot (u(t) - E_I).$$  

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 \( E_{\text{th}} \), the neuron will suddenly depolarise up to a value \( E_{\text{spike}} \), followed by a decrease below the resting potential \( E_L \) (hyperpolarisation) to the reset potential \( E_{\text{reset}} \). 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).

Mechanistically, this behaviour is produced by voltage-gated sodium and potassium ion channels (\( \text{Na}^+, \text{K}^+ \)): the probability of open \( \text{Na}^+ \) channels increases with the membrane potential, causing a positive feedback loop and a depolarisation of the membrane up to \( E_{\text{spike}} \). Meanwhile, the voltage-gated channels for \( \text{K}^+ \) open with the same mechanism, albeit a little slower, and the \( \text{Na}^+ \) channels transition into a closed and deactivated state, causing the sudden re- and hyperpolarisation. Facilitated by the hyperpolarisation the \( \text{Na}^+ \) and \( \text{K}^+ \) 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-
2.2 Biophysical Neuron Model

The signal has a finite 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].

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 2.10: Chemical synapse, adapted from https://commons.wikimedia.org/wiki/File:SynapseSchematic_en.svg. See text for description.
BACKGROUND AND RELATED WORK

Figure 2.11: Sketches of spiking neuron behavioural patterns. Each subgraph shows membrane potential $u(t)$ and input current $I_{syn}(t)$ 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].

### 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 $I_{\text{chan}}(u,t)$, corresponding to the sum of ionic currents through the ion channels, and the synaptic – or external – current $I_{\text{syn}}(u,t)$ modelling the ionic currents in the synapses as response to external input

$$-C_m \cdot \dot{u}(t) = i(t) = I_{\text{chan}}(u(t), t) + I_{\text{syn}}(u(t), t).$$  \hfill (2.6)

The form of the channel current $I_{\text{chan}}$ depends on the concrete neuron model, whereas the synaptic current $I_{\text{syn}}$ is determined by the synapse model.

### 2.3.2 Synapse models

Generally, the synaptic current $I_{\text{syn}}$ 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) :

$$I_{\text{syn}}(u, t) = \sum_k I_{\text{syn}}^k(u, t).$$  \hfill (2.7)

The current $I_{\text{syn}}^k$ 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 $I_{\text{syn}}^k$ itself. This model is particularly interesting as $I_{\text{syn}}^k$ 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

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.

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

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.
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 $\tau_k$, modelling the low-pass behaviour mentioned in Section 2.2.3. A single current based synapse $k$ can be described as

$$I^k_{\text{syn}}(t) \leftarrow I^k_{\text{syn}}(t) + w_k \quad \text{on spike for } k \text{ at } t$$  \hspace{1cm} (2.8)

$$-\frac{d}{dt}\tau_k \cdot I^k_{\text{syn}}(t) = I^k_{\text{syn}}(t)$$  \hspace{1cm} (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 $\tau_k$.

$$g_k(t) \leftarrow g_k(t) + w_k \quad \text{on spike for } k \text{ at } t$$  \hspace{1cm} (2.10)

$$-\frac{d}{dt}\tau_k \cdot g_k(t) = g_k(t)$$  \hspace{1cm} (2.11)

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

$$I^k_{\text{syn}}(u, t) = g_k(t) \cdot (u - E_k)$$  \hspace{1cm} (2.12)

Note that the dependency of $I^k_{\text{syn}}$ 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 theory, the reversal potential $E_k$ and time constant $\tau_k$ 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 $E_e$, $\tau_e$ (corresponding to the Na$^+$ ion channels), and an inhibitory synapse with parameters $E_i$, $\tau_i$ (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 $E_{\text{Th}}$ and enable the generation of output spikes. Analogously, the inhibitory synapse reversal
potential is chosen as \( E_i \leq E_L \), 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

\[
I_{syn}(u, t) = g_e(t) \cdot (u - E_e) + g_i(t) \cdot (u - E_i),
\]

(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 \( \tau_e \) and \( \tau_i \) 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

\[
-C_m \cdot \dot{u}(t) = g_L \cdot (u(t) - E_L) + I_{syn}(u(t), t)
\]

\[
= g_L \cdot (u(t) - E_L) + g_e(t) \cdot (u(t) - E_e) + g_i(t) \cdot (u(t) - E_i).
\]

(2.14)

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 \( E_{Th} \) > \( E_L \). The refractory period is modelled by tracking the last output spike time \( t_{spike} \) (initialised with \(-\infty\)). While the condition \( t - t_{spike} \leq \tau_{ref} \) holds, the membrane potential is clamped to the reset potential \( E_{reset} \leq E_L \)

\[
t_{spike} \leftarrow t \quad \text{if} \quad u(t) \geq E_{Th}
\]

\[
u(t) \leftarrow E_{reset} \quad \text{while} \quad t - t_{spike} \leq \tau_{ref}.
\]

(2.15) (2.16)

The expressiveness of this model is severely limited: given a constant input current \( I_{syn}(t) \), 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 \( E_L < u < E_{Th} \), given \( I_{syn} = 0 \), the differential \( \dot{u}(u) \) in Equation (2.14)
BACKGROUND AND RELATED WORK

(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].

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).

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

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

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

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

$$f_{QIF}(u) = -f_1 \cdot (u - f_2)^2 \quad f_{EIF}(u) = u - f_1 \cdot \exp(u - f_2) .$$

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 $E_{Th}'$, the potential is quickly pushed towards $E_{Th}$. Analogously to the LIF model, for $u < E_{Th}'$ 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 $E_{Th}(t)$. 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 the original AdEx paper and corresponding code, $E_{Th}^{exp}$ is usually referred to as $E_{Th}$, and $E_{Th}$ is replaced by a potential $E_{spike}$.

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

$$-C_m \cdot \dot{u}(t) = g_L \cdot (u(t) - E_L) + I_{Th}(u(t)) + I_a(t) + I_{syn}(u(t), t)$$  \hspace{1cm} (2.21)

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

$$I_{Th}(u) = g_L \cdot \Delta_{Th} \cdot \exp \left( \frac{u - E_{Th}^{exp}}{\Delta_{Th}} \right).$$  \hspace{1cm} (2.22)

The adaptation current $I_a(t)$ can be biologically interpreted as a form of habituation or short-time plasticity [Kan+12].
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 $\Delta t = 5$ 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 $E_{\text{Th}}$ 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 $I_{\text{Th}}$, which – depending on the implementation – can be achieved by setting $\Delta_{\text{Th}}$ to zero. Table 2.1 gives an overview of all parameters in the AdEx and LIF model with conductance based excitatory and inhibitory synapses.
## 2.3 Simplified Neuron and Synapse Models

<table>
<thead>
<tr>
<th>Potentials</th>
<th>LIF</th>
<th>ADEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_L$</td>
<td>-65.0</td>
<td>-70.6  [mV]</td>
</tr>
<tr>
<td>$E_{Th}$</td>
<td>-50.0</td>
<td>-40.0  [mV]</td>
</tr>
<tr>
<td>$E_{reset}$</td>
<td>-65.0</td>
<td>-70.6  [mV]</td>
</tr>
<tr>
<td>$E_e$</td>
<td>0.0</td>
<td>0.0    [mV]</td>
</tr>
<tr>
<td>$E_i$</td>
<td>-70.0</td>
<td>-80.0  [mV]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time constants</th>
<th>LIF</th>
<th>ADEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{ref}$</td>
<td>0.1</td>
<td>0.1    [ms]</td>
</tr>
<tr>
<td>$\tau_e$</td>
<td>5.0</td>
<td>5.0    [ms]</td>
</tr>
<tr>
<td>$\tau_i$</td>
<td>5.0</td>
<td>5.0    [ms]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Membrane parameters</th>
<th>LIF</th>
<th>ADEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_m$</td>
<td>1.0</td>
<td>0.281  [nF]</td>
</tr>
<tr>
<td>$g_L$</td>
<td>0.05</td>
<td>0.03    [µS]</td>
</tr>
<tr>
<td>$\tau_m$</td>
<td>20.0</td>
<td>9.37    [ms]</td>
</tr>
</tbody>
</table>

AdEx adaptation and exponential current mechanism

<table>
<thead>
<tr>
<th>Description</th>
<th>LIF</th>
<th>ADEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>/</td>
<td>4.0    [nS]</td>
</tr>
<tr>
<td>$b$</td>
<td>/</td>
<td>0.08   [nA]</td>
</tr>
<tr>
<td>$\tau_a$</td>
<td>/</td>
<td>144.0  [ms]</td>
</tr>
<tr>
<td>$\tau_{exp}^{-}\text{Th}$</td>
<td>/</td>
<td>-50.4  [mV]</td>
</tr>
<tr>
<td>$\Delta_{\text{Th}}$</td>
<td>/</td>
<td>2.0    [mV]</td>
</tr>
</tbody>
</table>

State variables

<table>
<thead>
<tr>
<th>Description</th>
<th>LIF</th>
<th>ADEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u(t)$</td>
<td>/</td>
<td>/     [V]</td>
</tr>
<tr>
<td>$I_a(t)$</td>
<td>/</td>
<td>/     [A]</td>
</tr>
<tr>
<td>$g_e(t)$</td>
<td>/</td>
<td>/     [S]</td>
</tr>
<tr>
<td>$g_i(t)$</td>
<td>/</td>
<td>/     [S]</td>
</tr>
</tbody>
</table>

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.

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 multi-processor 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,
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 $224 \times 256$ matrix of analogue synapse circuits. Adjacent rows in the synapse matrix receive external input via synapse drivers (112 per block), which are connected to
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.

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].

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 $\tau_{\text{ref}}$ and $g_L$ can be chosen per neuron, all other neuron parameters are shared by groups of 96 neurons [Pfe+13].
2.4 Neuromorphic Hardware

![Diagram](image)

**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 $64 \times 64$ 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 $\delta$

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

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 $(\vec{y}_k)_i$. Otherwise $w_{ij}$ is set to $w_{ij} = -1$

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

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).
2.5.2 Formal description of the Willshaw model

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 $M'$

$$M' = M \lor (\vec{x}_k \cdot \vec{y}_k^\top), \quad (2.25)$$

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

$$\mathcal{D} = \{(\vec{x}_k, \vec{y}_k) \mid k \in \{1, \ldots, N\}\}$$

with $\vec{x}_k = \vec{x}_\ell \iff k = \ell$ (2.26) is given in advance, a pre-calculated storage matrix $M$ can be obtained according to the following expression (Figure 2.17)

$$M = \bigvee_{j=1}^N \vec{x}_j \cdot \vec{y}_j^\top. \quad (2.27)$$
2.5 The Willshaw Associative Memory Model (BiNAM)

![Matrix Diagram](image)

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 $\tilde{y}_k^T = \vec{x}_k^T \cdot M$ before applying the threshold function $\Theta_c$. 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 $d$.

*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

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

(2.28)

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

$$(\Theta_\theta(\vec{z}))_i = H((\vec{z})_i - \theta).$$

(2.29)

Although the equation for $\Theta_\theta$ 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 $(\tilde{y}_k)_j$ was set to zero.

Generation of data $D$ as used in the experiments conducted in this thesis is discussed in detail in Section 3.3.
2.5.3 Choice of the threshold $\theta$

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

$$\vec{y} = (\vec{x}_k \cdot \vec{y}_k^\top)^\top \cdot \vec{x}_k = \vec{y}_k \cdot (\vec{x}_k^\top \cdot \vec{x}_k) = \vec{y}_k \cdot c.$$  (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 “$\lor$” in the training phase, newly trained samples can never lower the value of a single component $i$ in $\vec{y}$, but only cause an increase towards the upper limit $c$

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

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

$$M' = M \lor (\vec{x} \cdot \vec{y}^\top)$$

for any $\vec{x} \in \mathbb{B}^m, \vec{y} \in \mathbb{B}^n$. (2.32)

Given these considerations, adaptively setting the threshold $\theta$ 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]

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

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

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

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

For auto-associative storage measures see [Pal80].
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 $I = 3547$.

The 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:

$$I = N \cdot \log \left( \frac{n}{d} \right)$$  \hspace{1cm} (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 $n_{fp}^k$ for a specific sample $k$, the possible number of states occupied by those wrong bits needs to be subtracted in the information calculation:

$$I = \sum_{k=1}^{N} \log \left( \frac{n}{d} \right) - \log \left( \frac{d + n_{fp}^k}{d} \right)$$  \hspace{1cm} (2.36)

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

$$I = \sum_{k=1}^{N} \log \left( \frac{n}{d} \right) - \log \left( \frac{n_{fp}^k + d - n_{fn}^k}{d} \right) - \log \left( \frac{n - n_{fp}^k - d + n_{fn}^k}{n_{fn}^k} \right)$$  \hspace{1cm} (2.37)

As false positive and negative counts $n_{fp}^k$ and $n_{fn}^k$ 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 \( \mathbf{x} \), the \( \mathbf{y}_k \) 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 \( \mathbb{I} \) 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 \( \mathbf{x} \) are appointed to “column selectors” which contain a single “one”, corresponding to \( c = ||\mathbf{x}|| = 1 \). Let, without any loss of generality, assume that the \( \mathbf{x}_k \) are sorted such that \( (\mathbf{x}_k)_k = 1 \) and \( N = m \)

\[
M = \bigvee_{k=1}^{m} \mathbf{x}_k \cdot \mathbf{y}_k^\top = (\mathbf{y}_1, \ldots, \mathbf{y}_m)^\top \Rightarrow M^\top \cdot \mathbf{x}_k = \mathbf{y}_k. \tag{2.38}
\]

### 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 \( \theta \) is discussed in Section 2.5.6.

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 \( \mathbf{x} \), 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 $64 \times 64$ 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 $(M)_{ij} = 1$. The threshold $\theta$ 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 $\theta$ is compared. In the fixed threshold case $\theta = m = 8$.

in $\tilde{y}$. An excitatory synapse is added between input component $i$ and output neuron $j$ if, and only if, $(M)_{i,j} = 1$. The threshold $\theta$ 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 $\tilde{x}$.

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 $\tilde{x}_k$ presented to the network in the recall phase are the same as in the input data $\mathcal{D}$ 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 $1$ (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

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

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

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

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 $\theta$. 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 $\theta$. 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 $\theta$ 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.
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.

3.1 NEURAL NETWORK TOPOLOGY AND DATA CODING

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.

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.

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

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.

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).
3.1 Neural Network Topology and Data Encoding

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

<table>
<thead>
<tr>
<th>Classical network</th>
<th>Spiking network</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input vector</strong></td>
<td><strong>Spikes per neuron</strong></td>
</tr>
<tr>
<td>$\vec{x}_1$</td>
<td>$i = 1$</td>
</tr>
<tr>
<td>$\vec{x}_2$</td>
<td>$i = 2$</td>
</tr>
<tr>
<td>$\vec{x}_3$</td>
<td>$i = 3$</td>
</tr>
<tr>
<td>$\vec{x}_4$</td>
<td>$i = 4$</td>
</tr>
<tr>
<td>$\vec{x}_5$</td>
<td>$i = 5$</td>
</tr>
<tr>
<td><strong>Output vector</strong></td>
<td><strong>Continuous time $t$</strong></td>
</tr>
<tr>
<td>$\vec{y}_1$</td>
<td>$t = 1$</td>
</tr>
<tr>
<td>$\vec{y}_2$</td>
<td>$t = 2$</td>
</tr>
<tr>
<td>$\vec{y}_3$</td>
<td>$t = 3$</td>
</tr>
<tr>
<td>$\vec{y}_4$</td>
<td>$t = 4$</td>
</tr>
<tr>
<td>$\vec{y}_5$</td>
<td>$t = 5$</td>
</tr>
</tbody>
</table>

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 \( t_{i,k}^{\text{in}} \), which can be described as an ordered sequence of spike times \( t_{i,k}^{\text{in}} \)

\[
t_{i,k}^{\text{in}} = (t_{i,k,1}^{\text{in}}, t_{i,k,2}^{\text{in}}, \ldots) \quad \text{with} \quad t_{i,k,\ell}^{\text{in}} \leq t_{i,k,\ell'}^{\text{in}} \leftrightarrow \ell \leq \ell'. \tag{3.1}
\]

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

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

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

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 \( \zeta \) between the first input and output spike of a sample is known, the output spike train can be virtually shifted back by \( \zeta \) for the calculation of \( k_{j,k}^{\text{out}} \) according to Equation (3.2)

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

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.
3.1 Neural Network Topology and Data Encoding

![Diagram of neural network topology and data encoding]

Figure 3.4: Example of input sample pipelining: virtually shifting the output spike times back by \( \zeta \) 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 \( k_{j,\ell}^{\text{out}} = k \) 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 $s_{out} \geq 1$ 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.

When generating artificial bursts as test input, we parametrise these with an equidistant interspike interval (ISI) $\Delta t$. 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 $\Delta t$ 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 $(\vec{y}_k)_{j}^{\ell}$ as the quotient of the number of actual number of spikes encountered in the output time window, divided by the expected burst size $s_{out}$

$$
(\vec{y}_k)_{j}^{\ell} = \frac{|\{k_{out}^{\ell} = k\}|}{s_{out}}.
$$

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 $\mu_{i}\text{offs}$ (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 $\sigma_i$ for each spike time $t_{i\text{in}}$. Additionally, we shift the entire burst by $\mu_{i\text{offs}}$ in order to model synaptic or spatial delays. Again, $\mu_{i\text{offs}}$ is sampled from a Gaussian distribution with standard deviation $\sigma_{i\text{offs}}$.

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

$$t_i \sim \mathcal{N}\left(k \cdot T + \mu_{i\text{offs}} + \Delta t \cdot (\ell - 1), \sigma_i\right) \text{ with } \mu_{i\text{offs}} \sim \mathcal{N}(0, \sigma_{i\text{offs}}) \quad (3.5)$$

Note that the offset $\mu_{i\text{offs}}$ 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 $(\vec{x}_k)$ 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].

With respect to our scenario, the following changes to the network topology take these considerations into account: we replace each neu-
spiking associative memory architecture and testing

<table>
<thead>
<tr>
<th>Network and input/output parametrisation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network and data parameters</strong></td>
</tr>
<tr>
<td>( m, n )</td>
</tr>
<tr>
<td>( c, d )</td>
</tr>
<tr>
<td>( N )</td>
</tr>
<tr>
<td>( K )</td>
</tr>
</tbody>
</table>

**Output data parameters**

\( s_{\text{out}} \) | 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**

\( s_{\text{in}} \) | 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. |
| \( \Delta t \) | Equidistant interspike interval between input spikes in a burst. |

**Input data noise parameters**

\( \sigma_1 \) | Standard deviation of the Gaussian distribution the input spikes are selected from. |
| \( \sigma_{\text{offs}} \) | 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 \( u^1_j, \ldots, u^K_j \), in which each fires a small number of \( s_{\text{out}} \) spikes per one-bit in the output component \((\vec{y})_j \). These neurons receive the same input and all output signals \( y^1_j, \ldots, y^K_j \) 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

\[
(\vec{y}_k)_j = \frac{|\{ k_{i,t} = k \}|}{K \cdot s_{\text{out}}}. \tag{3.6}
\]

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
3.1 NEURAL NETWORK TOPOLOGY AND DATA ENCODING

Algorithm 3.1: Input spike train generation algorithm. Generates the input spike trains \( t^\text{in} \) and indices \( k^\text{in} \) 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 \( m \).

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 fulfill a thresholding behaviour: let \( n_{\text{in}} \) 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

\[
\begin{align*}
n_{\text{out}}(n_{\text{in}}) &= \begin{cases} 
s_{\text{out}} & \text{if } n_{\text{in}} \geq n_1 = c \cdot s_{\text{in}} \cdot K \\
0 & \text{if } n_{\text{in}} \leq n_0 = (c - 1) \cdot s_{\text{in}} \cdot K \end{cases}.
\end{align*}
\]

(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

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
3.2 MEMORY EVALUATION MEASURES

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

$$\text{lb} \left( \binom{n}{k} \right) = \text{lb} \left( \frac{n!}{n! \cdot (n-k)!} \right) = \text{lb} \left( \frac{\Gamma(n+1)}{\Gamma(k+1) \cdot \Gamma(n-k+1)} \right)$$

$$= \text{lb}(\Gamma(n+1)) - \text{lb}(\Gamma(k+1)) - \text{lb}(\Gamma(n-k+1)). \quad (3.8)$$

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 $(\vec{y}_k)_j$: 

$$n_k^{fp} = \sum_{j: (\hat{y}_k)_j = 0} \min \{1, (\hat{y}_k)_j \} \quad n_k^{fn} = \sum_{j: (\hat{y}_k)_j = 1} 1 - \min \{1, (\hat{y}_k)_j \} \quad (3.9)$$

With these adaptations the storage capacity measure can be used in conjunction with spiking BiNAM implementations. In case the neurons fulfill 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 $\eta$ on another evaluation measure – for example the storage capacity – under variation of $\eta$. 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 $\sigma_t$ and $\sigma_{t\text{of}}$, 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.

**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 $\eta$ from a Gaussian distribution with standard deviation $\sigma_\phi$ and add it to the given value for the parameter $\phi$. 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
3.2 MEMORY EVALUATION MEASURES

throughput describes the rate new operations can be issued at.

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

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

(3.10)

The latency for a single sample $\delta_k$ is now defined as

$$\delta_k = \hat{t}_k^{out} - \hat{t}_k^{in} .$$

(3.11)

The above definition has two problems. $\delta$ 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 $\delta$, 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 $\delta$ 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 $T_p^{\text{crit}}$. 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.

Certain memory usage patterns might cause interference, which causes the value of $T_p^{\text{crit}}$ 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

$$J = \int_{t=0}^{\infty} (E_L - u(t)) \cdot i(t) \, dt ,$$

(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 $\mathcal{D} = \{ (\bar{x}_k, \bar{y}_k) \}$ without duplicate input vectors $\bar{x}_k$ 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: $\|\bar{x}_k\|_1 = c$ and $\|\bar{y}_k\|_1 = d$. 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
3.3 Data Generation

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

$$ (M^T \cdot \vec{x}_k)_j = \sum_{i : (\vec{x}_k)_i = 1} (M)_{ij} = \|\vec{x}_k\|_1 = c. \quad (3.13) $$

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 $\langle n_{fp} \rangle$ can be easily calculated.

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

$$ \varphi = \Pr((M)_{ij} = 1) = 1 - \Pr((M)_{ij} = 0) \quad (3.14) $$

$$ = 1 - \prod_{k=1}^{N} \Pr((\vec{x}_k)_i = 0 \lor (\vec{y}_k)_j = 0) \quad (3.15) $$

$$ = 1 - \prod_{k=1}^{N} \left( 1 - \Pr((\vec{x}_k)_i = 1 \land (\vec{y}_k)_j = 1) \right) \quad (3.16) $$

$$ = 1 - \prod_{k=1}^{N} \left( 1 - \Pr((\vec{x}_k)_i = 1) \cdot \Pr((\vec{y}_k)_j = 1) \right) \quad (3.17) $$

$$ = 1 - \left( 1 - \frac{c \cdot d}{m \cdot n} \right)^N. \quad (3.18) $$

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

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

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

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 “$\lor$”-operation of Equation (3.20) without changing its value

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

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

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 $\Pr((M)_{ij} = 0)$ as a product: the individual events are no longer independent. However, the error introduced by this assumption is sufficiently small for large memories [Pal80].

53
Now it is clearly visible that there is no dependence between \( M \) and \( \bar{x}_k \) in Equation (3.19). The final \( \Pr(C(k,j)) \) is thus given as

\[
\Pr(C(k,j)) = \Pr\left( \bigwedge_{i=1}^{n} (M)_{ij} = 1 \right) = \varphi^c = \left( 1 - \left( 1 - \frac{c \cdot d}{m \cdot n} \right)^N \right)^c. \tag{3.22}
\]

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

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

Figure 3.8 shows \( \langle n_{fp} \rangle \) 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 \( B = (\bar{b}_1, \ldots, \bar{b}_N)^\top \), where each \( \bar{b}_i \) is an \( r \)-tuple containing \( r_1 \) ones and \( r_0 = r - r_1 \) zeros

\[
\bar{b}_i \in \mathcal{B}_{r_1} = \left\{ (b_1, \ldots, b_r) \mid b_j \in \mathbb{B}, \sum_{j=1}^{r} b_j = r_1 \right\}. \tag{3.24}
\]

According to basic combinatorics the size of \( \mathcal{B}_{r_1} \) is

\[
|\mathcal{B}_{r_1}| = \binom{r}{r_1} = \binom{r}{r - r_1} = \binom{r}{r_0}. \tag{3.25}
\]
3.3 Data Generation

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 $\vec{b} \in B_{r_1}$ can be uniquely represented as set $b$ containing the indices of “ones” in $\vec{b}$ (Figure 3.9)

$$b = \{i \mid b_i = 1\} \quad b \in B_{r_1} = \{\{i_1, \ldots, i_{r_1}\} \mid i_j \in \{1, \ldots, r\}\}. \quad (3.26)$$

The set $B_{r_1}$ 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 $O(n \cdot r_1)$ 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 3.9: Elements of $B^5_3$ and their corresponding set representation.

$(1,0,1,0,0) \rightarrow \{0,2\}$
$(0,0,1,0,1) \rightarrow \{2,4\}$
$(0,1,0,0,1) \rightarrow \{1,5\}$
spiking associative memory architecture and testing

Random
(with duplicates)
Random
(no duplicates)
Balanced
(with duplicates)
Balanced
(no duplicates)

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 $32 \times 32$ 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

$$\max_j \left\{ \sum_{k'=1}^{r'} B_{kj} \right\} - \min_j \left\{ \sum_{k'=1}^{r'} B_{kj} \right\} \leq 1 \forall k' \leq N . \quad (3.27)$$

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.

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).

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.
Figure 3.12: Random versus balanced data generation. 1000 trials in which a $16 \times 16$ 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.
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 $\vec{b}_k$ 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 $i_j > i_{j'} \iff j' > j$. 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.

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

$$N \rightarrow j = |\vec{b}_j| = \begin{cases} \binom{j}{\tilde{r}} & \text{if } j \geq \tilde{r} \\ 0 & \text{if } j < \tilde{r} \end{cases}.$$  \hspace{1cm} (3.28)

Whenever a path along a child node $\aleph[j]$ is traversed, the corresponding table entry $N \rightarrow j$ is decremented, as a new combination with this path as prefix is being generated. Once $N \rightarrow j$ 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.
3.3 Data Generation

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.

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 $\mathbb{N}^{-j}$ upon first access. See text for description.
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.

An important detail is to bias the index selection by the number of possible paths continuing with \( j, N^{-j} \). 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.

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 \( s \) to those for which the usage count is minimal

\[
\mathbf{s}_{\text{min}} = s \cap \{ j \mid (\vec{u})_j = \min\{(\vec{u})_{j'} \mid j' \in s\} \} \quad \text{where } s = \{ j \mid N^{-j} > 0 \}.
\]

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
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 $s_{\text{bal}}$ which allow the selection of at least $\bar{r} = r_1 - ℓ$ remaining ones with minimal usage

$$s_{\text{bal}} = \left\{ j \mid \sum_{j' = 1}^{j} \max\{0, 1 - (\vec{u})_{j'} + \min(\vec{u})\} \geq \bar{r}, j \in \{1, \ldots, i - 1\} \right\}.$$  

The final selectable indices $s_{\text{sel}}$ are then given as

$$s_{\text{sel}} = \begin{cases} s_{\text{min}} \cap s_{\text{bal}} & \text{if } s_{\text{min}} \cap s_{\text{bal}} \neq \emptyset \\ s_{\text{min}} & \text{otherwise} \end{cases}.$$  

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 $O(N \cdot r_1 \cdot r)$ the runtime of the method linearly depends on $N$ and $r$, which is worse than the previously presented $O(N \cdot r_1)$ algorithm for non-balanced, non-unique samples.

3.4 CONCLUSION

This chapter outlined the design space of spiking BiNAM implementations: apart from 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 $\Phi$ for the network parameters listed above. We then move to the actual full network experiments on neuromorphic hardware in Chapter 5.
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 $\Phi$ 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 $\Phi$ 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
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 $\Phi$ 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 $\Phi$ 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.

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].

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 $(\mathbf{y}_i)$, 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
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 $\sigma_t$. 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 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 $\sigma_w$.

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).

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 $\Phi$, one excitatory synapse with adaptive weight and an according input spike train $(t^{in}, w^{in})$ 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 $O(1)$ of single neuron simulation compared to the simulation of a full network in $O(n)$.

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

\[
C_m > 0, g_L > 0, \tau_e > 0, \tau_i > 0, \tau_a > 0, \tau_{ref} \geq 0, w \geq 0, a \geq 0. \tag{4.1}
\]

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

\[
E_{Th} > E_e > E_{Th}^{exp} > E_L \geq E_i \geq E_{reset}, \tag{4.2}
\]

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

\[
E_e > E_{Th} > E_L \geq E_i \geq E_{reset}. \tag{4.3}
\]

The exponential slope \(\Delta_{Th}\) and the adaptation current \(b\) are subject to the constraints

\[
\Delta_{Th} < E_{Th}^{exp} - E_L, b \geq 0. \tag{4.4}
\]
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.

**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 $\tau$ into decay rates $\lambda$. 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.
4.1 Design Space Exploration

### Reduced DOF AdEx and LIF Parameters and Neuron State

#### Potentials

<table>
<thead>
<tr>
<th>UNIT</th>
<th>DESCRIPTION</th>
<th>TRANSFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E_{\text{Th}'}) [V]</td>
<td>Threshold potential</td>
<td>(E_{\text{Th}} - E_L)</td>
</tr>
<tr>
<td>(E_{\text{exp}'}) [V]</td>
<td>Spike potential</td>
<td>(E_{\text{exp}} - E_L)</td>
</tr>
<tr>
<td>(E_{\text{reset}'}) [V]</td>
<td>Reset potential</td>
<td>(E_{\text{reset}} - E_L)</td>
</tr>
<tr>
<td>(E_e') [V]</td>
<td>Excitatory reversal potential</td>
<td>(E_e - E_L)</td>
</tr>
<tr>
<td>(E_i') [V]</td>
<td>Inhibitory reversal potential</td>
<td>(E_i - E_L)</td>
</tr>
</tbody>
</table>

#### Decay rates

<table>
<thead>
<tr>
<th>UNIT</th>
<th>DESCRIPTION</th>
<th>TRANSFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_L) [s(^{-1})]</td>
<td>Membrane leak rate</td>
<td>(g_L/C_m)</td>
</tr>
<tr>
<td>(\lambda_a) [s(^{-1})]</td>
<td>Adaptation decay rate</td>
<td>(1/\tau_a)</td>
</tr>
<tr>
<td>(\lambda_e) [s(^{-1})]</td>
<td>Excitatory channel decay rate</td>
<td>(1/\tau_e)</td>
</tr>
<tr>
<td>(\lambda_i) [s(^{-1})]</td>
<td>Inhibitory channel decay rate</td>
<td>(1/\tau_i)</td>
</tr>
</tbody>
</table>

#### Other parameters

<table>
<thead>
<tr>
<th>UNIT</th>
<th>DESCRIPTION</th>
<th>TRANSFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_w) [s(^{-1})]</td>
<td>Synapse weight</td>
<td>(w/C_m)</td>
</tr>
<tr>
<td>(f_a) [s(^{-1})]</td>
<td>Subthreshold adaptation</td>
<td>(a/C_m)</td>
</tr>
<tr>
<td>(\Delta_b) [V s(^{-1})]</td>
<td>Spike-triggered adaptation</td>
<td>(b/C_m)</td>
</tr>
<tr>
<td>(\tau_{\text{ref}}) [s]</td>
<td>Refractory period</td>
<td>(\tau_{\text{ref}})</td>
</tr>
<tr>
<td>(\Delta_{\text{Th}}) [V]</td>
<td>Spike slope factor</td>
<td>(\Delta_{\text{Th}})</td>
</tr>
</tbody>
</table>

#### State variables

<table>
<thead>
<tr>
<th>UNIT</th>
<th>DESCRIPTION</th>
<th>TRANSFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u'(t)) [V]</td>
<td>Membrane potential</td>
<td>(u(t) - E_L)</td>
</tr>
<tr>
<td>(\Delta_{\text{a}}(t)) [V s(^{-1})]</td>
<td>Adaptation current</td>
<td>(l_{\text{a}}(t)/C_m)</td>
</tr>
<tr>
<td>(f_e(t)) [s(^{-1})]</td>
<td>Excitatory channel frequency</td>
<td>(g_e(t)/C_m)</td>
</tr>
<tr>
<td>(f_i(t)) [s(^{-1})]</td>
<td>Inhibitory channel frequency</td>
<td>(g_i(t)/C_m)</td>
</tr>
</tbody>
</table>

*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 $t^{\text{in}}$ with annotated weights $w^{\text{in}}$ 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 $t^{\text{in}}$ 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.

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

$$\frac{d}{dt} \vec{v}(t) = f(\vec{v}(t)) .$$

(4.5)

The evaluation of autonomous systems is potentially faster than that of equivalent non-autonomous systems $f(t, \vec{v}(t))$. 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
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 $t_{\text{spike}}$ is the time of the last output spike. Furthermore, in the special case of single neuron simulation, the input spike times $t^{\text{in}}$ 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,

$$
\begin{align*}
\frac{d}{dt}u &= - E_{\text{reset}} + \sum_{i} w^{\text{in}}[i] \delta(t - t^{\text{in}}[i]) \\
\frac{d}{dt}h &= \frac{1}{\tau_{\text{ref}}} (E_{\text{out}} - h) \\
\frac{d}{dt}a &= \begin{cases} 
\frac{1}{\Delta_{a}} (E_{\text{reset}} - a) & \text{if } t < t^{\text{in}}[i] \\
0 & \text{otherwise}
\end{cases}
\end{align*}
$$

of which one is the refractory state, during which the differential $\frac{d}{dt}u$ must be set to zero.

Implementations of the $\text{Integrate}$ function are discussed in Section 4.2.3.

Algorithm 4.1: Basic single neuron simulator loop. Given an input spike time sequence $t^{\text{in}}$ with annotated weights $w^{\text{in}}$, the algorithm calculates output spike times $t^{\text{out}}$ of a single AdEx neuron. The system of differential equations is described in the functional $f$. $\text{inRef}$ specifies whether the neuron is in the refractory state, during which the differential $\frac{d}{dt}u$ must be set to zero.
if the integrator stops at these state changes boundaries. This can be accomplished by setting the maximum integrator timestep \( h \) to \( t_{\text{max}} - t \), where \( t_{\text{max}} \) 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.

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 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 \( I_{\text{Th}} \) which rises exponentially with the membrane potential \( u \) (Equation (2.21)):

\[
I_{\text{Th}} = g_L \cdot \Delta_{\text{Th}} \cdot \exp \left( (u - E_{\text{Th}}^{\text{exp}}) \cdot \Delta_{\text{Th}}^{-1} \right) \tag{4.6}
\]

In a multi-step numerical differential equation integrator, naive integration of \( I_{\text{Th}} \) may cause undesired side effects. Due to the exponential, \( I_{\text{Th}} \) reaches extreme values for membrane potentials \( u > E_{\text{Th}}^{\text{exp}} \). Such potentials may occur in a multi-step differential equation integrator, since the non-differentiable membrane potential reset on \( u > E_{\text{Th}} \) 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 \( I_{\text{Th}} \) to a reasonable maximum \( I_{\text{Th}}^{\text{max}} \) which (under the assumption of no other current) safely allows to cross the entire membrane potential dynamic range \( E_{\text{dyn}} = E_{\text{Th}} - E_{\text{reset}} \) in one timestep \( h \), but prevents an uncontrolled exponential avalanche. \( I_{\text{Th}}^{\text{max}} \) is given as:

\[
h \cdot \dot{u}(t) = h \cdot \frac{g_L \cdot I_{\text{Th}}^{\text{max}}}{C_m} \leq E_{\text{dyn}} \Leftrightarrow I_{\text{Th}}^{\text{max}} \leq E_{\text{dyn}} \cdot \frac{C_m}{g_L \cdot h}. \tag{4.7}
\]

The actual threshold current used in the integrator \( I_{\text{Th}}' \) is then defined as \( I_{\text{Th}}' = \min \{ I_{\text{Th}}^{\text{max}}, I_{\text{Th}} \} \).
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

\[ e^x = 2^x = 2^a \cdot 2^b \text{ with } x' = \ln(2) \cdot x, a = \lfloor x' \rfloor, b = x' - a, \]  

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

\[ \vec{v}(t) = \int_0^t f(\vec{v}(t)) \, dt + \vec{v}_0 \quad \text{with} \quad f(\vec{v}(t)) = \frac{d\vec{v}(t)}{dt} \]  

where \(\vec{v}_0 \in \mathbb{R}^\alpha\) denotes the initial state of the neuron, and the functional \(f : \mathbb{R}^\alpha \rightarrow \mathbb{R}^\alpha\) 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 \(\vec{v}'\):

\[ E(t) = \|\vec{v}(t) - \vec{v}'(t)\| \]  

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

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 \(\vec{v}'\):

\[ \vec{v}'(t + h) = \text{Integrate}(f, \vec{v}(t), h) \]  

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

\[ \vec{v}'(t + h) = \vec{v}(t) + h \cdot f(\vec{v}(t)). \]
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]

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

Using the coefficients \( b_j \) and \( a_{ij} \) 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 \( h_{\text{max}} \) and the target error \( e \). The actual step size \( h' \) is returned and must be passed to the next iteration

\[
h', \vec{v}'(t + h') = \text{INTEGRATE}(f, \vec{v}(t), h, h_{\text{max}}, e)
\] (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

\[
h' = h \cdot \left( \frac{1}{E} \right)^{1/5},
\] (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.
### 4.2 SINGLE NEURON SIMULATION

#### NEURON SIMULATOR BENCHMARK RESULTS

<table>
<thead>
<tr>
<th>Integrator</th>
<th>LIF</th>
<th>AdEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h = 1 \mu s$</td>
<td>2071.354</td>
<td>0.223</td>
</tr>
<tr>
<td>$h = 10 \mu s$</td>
<td>130.804</td>
<td>0.739</td>
</tr>
<tr>
<td>$h = 100 \mu s$</td>
<td>12.139</td>
<td>2.508</td>
</tr>
<tr>
<td>$h = 1 ms$</td>
<td>1.198</td>
<td>8.079</td>
</tr>
<tr>
<td>Midpoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h = 1 \mu s$</td>
<td>2412.932</td>
<td>0.033</td>
</tr>
<tr>
<td>$h = 10 \mu s$</td>
<td>165.374</td>
<td>0.948</td>
</tr>
<tr>
<td>$h = 100 \mu s$</td>
<td>13.282</td>
<td>3.156</td>
</tr>
<tr>
<td>$h = 1 ms$</td>
<td>1.482</td>
<td>9.820</td>
</tr>
<tr>
<td>Runge-Kutta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h = 1 \mu s$</td>
<td>3873.598</td>
<td>0.033</td>
</tr>
<tr>
<td>$h = 10 \mu s$</td>
<td>314.017</td>
<td>0.948</td>
</tr>
<tr>
<td>$h = 100 \mu s$</td>
<td>32.832</td>
<td>3.155</td>
</tr>
<tr>
<td>$h = 1 ms$</td>
<td>3.024</td>
<td>9.317</td>
</tr>
<tr>
<td>Dormand-Prince</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e = 1 \mu$</td>
<td>2286.730</td>
<td>0.033</td>
</tr>
<tr>
<td>$e = 10 \mu$</td>
<td>572.327</td>
<td>0.336</td>
</tr>
<tr>
<td>$e = 100 \mu$</td>
<td><strong>54.232</strong></td>
<td><strong>1.330</strong></td>
</tr>
<tr>
<td>$e = 1 m$</td>
<td>7.367</td>
<td>4.200</td>
</tr>
<tr>
<td>$e = 10 m$</td>
<td>2.621</td>
<td>10.879</td>
</tr>
<tr>
<td>$e = 100 m$</td>
<td>3.019</td>
<td>13.645</td>
</tr>
</tbody>
</table>

Table 4.4: Neuron simulator benchmark results. The column $\Delta u$ 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 $\Delta u$. 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 $h = 1$ 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 $P$ to a parameter vector $\Phi$. 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.
4.3 Approach 1: Spike Train

Figure 4.3: Example of a spike train evaluation. Bold vertical lines represent single input/output spikes. Row la 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 lb 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 \( \Phi \) 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 \( \Phi \). 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 \( P_{st} \) 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 \( \tilde{n}^{out} \). Individual input bursts are generated...
neuron parameter evaluation and optimisation

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 \( \sigma_t \), \( \sigma_{offs} \) and \( \Delta t \). The annotated weights in \( w_{in} \) are selected according to the specified synapse weight \( w \) with superimposed Gaussian noise according to the noise parameter \( \sigma_w \). 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.

4.3.3 Evaluation

Given the input \((t_{in}, w_{in})\), the behaviour of a single neuron with parameters \( \Phi \) is simulated and the output spike train \( t_{out} \) is recorded. Let \( \tilde{n}_{out} \) denote the expected number of output spikes for group \( i \) and

\[
\tilde{n}^{out}_{i} = |\{ t_{j}^{out} | T \cdot (i - 1) \leq t_{j}^{out} < T \cdot i \}|
\]  (4.16)
4.4 APPROACH 2: SINGLE GROUP, SINGLE OUTPUT SPIKE

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

\[
P_{st}^i(\Phi) = \begin{cases} 
1 & \text{if } n_{i_{\text{out}}}^i = n_{i_{\text{out}}}^i(\Phi) \\
0 & \text{otherwise}
\end{cases} 
\]

\[
P_{st}(\Phi) = \frac{1}{n_g} \sum_{i=1}^{n_g} P_{st}^i(\Phi) 
\]

For large \( n_g \) the target function \( P_{st}(\Phi) \) can be interpreted as the probability of the neuron with parameters \( \Phi \) 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 \( P_{st} \) has major drawbacks: \( P_{st}(n_g) \) 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 \( P_{sgso} \) 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 \( P_{sgso} \), and finally focus on the notion of the “effective threshold potential” \( E_{\text{Th}}^{\text{eff}} \).

4.4.1 Concept

The central idea of the SGSO measure is to analyse the distance between the maximum neuron membrane potential \( u_{\text{max}} \) and the effective threshold potential \( E_{\text{Th}}^{\text{eff}} \). 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 \( u_{\text{max}} \) reaches \( E_{\text{Th}}^{\text{eff}} \) 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 \( u_{\text{max}} \) and \( E_{\text{Th}}^{\text{eff}} \) 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
the time $T$ and comparing the result to the initial neuron state. The method is sketched in Figure 4.4.

A major challenge of this methodology is that – by definition – the neuron issues an output spike as soon as $E_{\text{Th}}$ is reached. As a result, it would hold $u_{\text{max}} = E_{\text{Th}}$. 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 $E_{\text{Th}}$ is reached. For the AdEx model, the exponential spike generation current must furthermore be limited to

$$I_{\text{Th}}'(u) = \min\{I_{\text{Th}}(u), I_{\text{Th}}(E_{\text{eff}}^{\text{Th}})\}.$$  \hspace{1cm} (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 $\sigma_t$ and $\sigma_{\text{offs}}$. As shown in Figure 4.5(a), determinism could be enforced by setting the spike time noise parameters $\sigma_t$ and $\sigma_{\text{offs}}$ 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 $\sigma_t$- and $\sigma_{\text{offs}}$-induced spread of spike times encountered when fusing multiple

---

**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 $u_{\text{max}}$ to the effective threshold potential $E_{\text{Th}}$. The neuron state at time $T$ determines how well it fulfils the reset condition.

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 satisfactory smooth measure.
4.4 APPROACH 2: SINGLE GROUP, SINGLE OUTPUT SPIKE

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 $\Delta t$. The $i$-th spike of each burst is at exactly the same time. (b) shows bursts randomly generated according to the parameters $\sigma_t$ and $\sigma_{\text{offs}}$ 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 $t_{\text{offs}}$.

4.4.3 Evaluation measure

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

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

An important property of $L(x \mid x_0)$ 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 $\tau$ in Equation (4.21) controls the slope of sigmoid function and consequently the “softness” of $p_{\text{sd}}$. $\tau$ is chosen according to

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

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

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 $\Delta t$. The $i$-th spike of each burst is at exactly the same time. (b) shows bursts randomly generated according to the parameters $\sigma_t$ and $\sigma_{\text{offs}}$ 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 $t_{\text{offs}}$. 

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 $t_{\text{offs}}$ to each burst in the input spike train. The offset is chosen as

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

where the denominator $n_{\text{bursts}}$ is the number of input bursts (either $c$ or $c - 1$), and the numerator $2 \cdot (\sigma_t + \sigma_{\text{offs}})$ the width of the 68.2%-quantile of the Gaussian distribution resulting from additive superposition of the Gaussian distributions accounting for $\sigma_t$ and $\sigma_{\text{offs}}$. Figure 4.5(c) sketches the effect of the equidistant offset.
plots of $L(x \mid x_0)$ are shown in Figure 4.6(a). The first ingredient to the evaluation measure, $p_{\text{th}}$, describes accordance with the threshold condition

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

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 $u^\text{reset}_{\max}$ result in a high valuation in the compound measure $p_{\text{th}}$.

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 $p_{\text{reset}}$ is measured as

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

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 $\Phi$ is defined as

$$\mathcal{P}_{\text{sgso}}(\Phi) = p_{\text{th}}(\Phi) \cdot p_{\text{reset}}(\Phi).$$
4.4 Approach 2: Single Group, Single Output Spike

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 \( I_{\text{th}} \) in the AdEx model (Equation (2.22))

\[
I_{\text{th}} = g_L \Delta_T \exp \left( (u(t) - E_{\text{th}}^{\exp}) \Delta_{\text{th}}^{-1} \right). \tag{4.25}
\]

For an absence of inhibitory currents \( I_h(t) \leq 0 \) and \( g_l(t) = 0 \), and a leak potential \( E_L = 0 \), \( E_{\text{th}}^{\text{eff}} \) is given as the membrane potential \( x \) at which leak and threshold current cancel each other out

\[
g_L x = g_L \Delta_{\text{th}} \exp \left( (x - E_{\text{th}}^{\exp}) \Delta_{\text{th}}^{-1} \right). \tag{4.26}
\]

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

**Lambert W function** Rearranging Equation (4.26) yields

\[
\Delta_{\text{th}} = x \exp \left( -(x - E_{\text{th}}^{\exp}) \Delta_{\text{th}}^{-1} \right)
\]

\[
\Leftrightarrow \exp \left( -E_{\text{th}}^{\exp} \Delta_{\text{th}}^{-1} \right) = -x \Delta_{\text{th}}^{-1} \exp \left( -x \Delta_{\text{th}}^{-1} \right). \tag{4.27}
\]

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

\[
E_{\text{th}}^{\text{eff}} = x = -\Delta_{\text{th}} \exp \left( -E_{\text{th}}^{\exp} \Delta_{\text{th}}^{-1} \right). \tag{4.28}
\]

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

\[
\exp \left( -E_{\text{th}}^{\text{eff}} \Delta_{\text{th}}^{-1} \right) < \exp(-1) \Leftrightarrow E_{\text{th}}^{\text{eff}} > \Delta_{\text{th}}. \tag{4.29}
\]

For \( E_{\text{th}}^{\text{eff}} \leq \Delta_{\text{th}} \) the threshold current \( I_{\text{th}} \) 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

\[
f(x) = 0 = \log (\Delta_{\text{th}}) + (x - E_{\text{th}}) \Delta_{\text{th}}^{-1} - \log (x). \tag{4.30}
\]

The iterative solution is given as

\[
x_0 = E_{\text{Th}},
\]

\[
x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{\log (\Delta_{\text{th}}) + (x - E_{\text{th}}) \Delta_{\text{th}}^{-1} - \log (x)}{\Delta_{\text{th}}^{-1} - x^{-1}},
\]

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 $q_{\text{out}}$. An efficient algorithm for the calculation of the said $q_{\text{out}}$ 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

$$E(\Phi) = |\hat{n}_{\text{out}} - n_{\text{out}}(\Phi, t^{\text{in}}_1)| + n_{\text{out}}(\Phi, t^{\text{in}}_0),$$

where $t^{\text{in}}_1$ and $t^{\text{in}}_0$ are deterministic input spike trains (Section 4.4.2) for which the neuron is expected to produce $\hat{n}_{\text{out}}$ and zero output spikes respectively. An eminent disadvantage of the above equation are the discrete spike counts $n_{\text{out}}$, which cause the error function $E(\Phi)$ to take the form of a step function.

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

$$E(\Phi) = |o + \hat{n}_{\text{out}} - q_{\text{out}}(\Phi, t^{\text{in}}_1)| + q_{\text{out}}(\Phi, t^{\text{in}}_0),$$

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 $\hat{n}_{\text{out}} - 1$ spikes, but also not too close to issuing $\hat{n}_{\text{out}} + 1$ spikes. In case $q_{\text{out}}$ 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 $o = 0.3$. 
4.5 Approach 3: Single Group, Multiple Output Spikes

Figure 4.7: Idealised sketch of the SGMO measure. The output spike count $n_{\text{out}}(\phi)$ over a neuron parameter $\phi$ 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 $q_{\text{out}}$ is to $n_{\text{out}}$ for input $t_{\text{in}}^1$ (sketched in Figure 4.7) and how close to zero for input $t_{\text{in}}^0$

$$p_{\text{th}}(\Phi) = \frac{1}{1 + (0 + n_{\text{out}} - q_{\text{out}}(\Phi, t_{\text{in}}^1))^2} \cdot \frac{1}{1 + (q_{\text{out}}(\Phi, t_{\text{in}}^0))^2}. \quad (4.34)$$

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

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

where $\vec{v}_T$ is the state of the neuron at time $T$ for input $t_{\text{in}}^0$ 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 $P_{\text{sgmo}}(\Phi)$ is given as

$$P_{\text{sgmo}}(\Phi) = p_{\text{th}}(\Phi) \cdot p_{\text{reset}}(\Phi). \quad (4.36)$$
4.5.2 Fractional spike count

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.

**Properties of the fractional spike count** For the sake of simplicity, let us consider a single variable parameter dimension with value \( \phi \). All other parameters in the parameter vector \( \Phi \) 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

\[
q^{\text{out}}(\phi) = n^{\text{out}}(\phi) + p^{\text{out}}(\phi) .
\]

(4.37)

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

\[
p^{\text{out}}(\phi) = 0 \Leftrightarrow n^{\text{out}}(\phi) > n^{\text{out}}(\phi - \epsilon) \lor n^{\text{out}}(\phi) > n^{\text{out}}(\phi + \epsilon), \quad (4.38)
\]

\[
p^{\text{out}}(\phi) \to 1 \Leftrightarrow n^{\text{out}}(\phi) < n^{\text{out}}(\phi - \epsilon) \lor n^{\text{out}}(\phi) < n^{\text{out}}(\phi + \epsilon), \quad (4.39)
\]

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

\[
n^{\text{out}}(\phi) = n^{\text{out}}(\phi') \quad \text{where} \quad \hat{\phi} \leq \phi, \phi' \leq \hat{\phi} .
\]

(4.40)

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

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
4.5 Approach 3: Single Group, Multiple Output Spikes

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

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

Repeatedly comparing the membrane potential traces $u(t)$ for such parameter pairs $\phi$ and $\phi'$ 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 $\phi'$. 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 $p^{\text{out}}$ could be defined as follows

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

which expresses the normalised minimal voltage difference between the local maxima of the membrane potential and the effective threshold potential $E^{\text{eff}}_{\text{Th}}$ (Section 4.4.4). If the maximum membrane potential is close to $E^{\text{eff}}_{\text{Th}}$, 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 $p^{\text{out}} = 0$.

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 $\phi'$ might be a necessary precondition for the production of a spike for $\phi$ 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 $\phi$. The most spectacular failure of this assumption arises if variation of $\phi$ indeed produces a spike at the predicted location, but in response a spike at another location disappears.

The normalisation of $p^{\text{out}}$ poses another problem. For the parameter value $\phi$, at which a new spike has just been introduced, Equation (4.38) requires $p^{\text{out}}$ to be set to zero. In practice the largest local maximum is larger than $E_{L}$, causing $p^{\text{out}}$ 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 $p^{\text{out}}$ 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.
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), $p_{\text{out}}$ 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 $p_{\text{out}}$ is caused by the second spike “jumping” to the position of the largest maximum (blue circles).
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 \( \Delta u \). The basic idea of the approach to the fractional spike count measure presented here is to find the minimal \( \Delta u \), 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 \( \Delta u \)?

Searching a minimal \( \Delta u \) 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 \( E_{\text{Th}}^{\text{eff}} \). It must hold \( \Delta u(t) \in [0, E_{\text{Th}}^{\text{eff}} - u(t)] \). Given these considerations, the entire fractional spike count calculation can be expressed mathematically as

\[
p_{\text{out}}(\Phi) = 1 - \min \left\{ \frac{\Delta u}{E_{\text{Th}}^{\text{eff}} - u(t + \tau_{\text{ref}})} \left| n_{\text{out}}^{\text{pert}}(\Phi, t + \tau_{\text{ref}}, \Delta u) > n_{\text{out}}^{\text{out}}(\Phi), \Delta u \in [0, E_{\text{Th}}^{\text{eff}} - u(t + \tau_{\text{ref}})], t \in (-\tau_{\text{ref}}) \parallel t_{\text{out}} \right\}, \quad (4.43)
\]

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

Algorithmic implementations of Equation (4.43) should take various optimisations into account. Most importantly, the search for the minimum \( \Delta u \) 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 \( \vec{\nu} \) 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 $n_{out} > 0$. For $n_{out} = 0$ $p_{out}$ (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 $p_{out}'$ in the figure) in regions with $n_{out} = 0$. 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, excluding 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.
Figure 4.10: Minimal membrane potential perturbation measure examples, comparison between LIF (left) and AdEx (right) neurons with sweeps over different parameter dimensions.
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.

---

**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 Neuron Evaluation Software Framework

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.
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 $\Phi$. 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 $P_{sgso}$ and $P_{sgmo}$...
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 $E_{\text{Th}}$.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Synapse</th>
<th>Membrane (LIF)</th>
<th>AdEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>0.03 µS</td>
<td>1.00 nF</td>
<td>4.00 nS</td>
</tr>
<tr>
<td>$E_e$</td>
<td>0.00 mV</td>
<td>0.08 µS</td>
<td>80.50 pA</td>
</tr>
<tr>
<td>$\tau_e$</td>
<td>5.00 ms</td>
<td>-70.00 mV</td>
<td>144.00 ms</td>
</tr>
<tr>
<td>$g_L$</td>
<td></td>
<td>-54.00 mV</td>
<td>20.00 mV</td>
</tr>
<tr>
<td>$E_{\text{Th}}$</td>
<td></td>
<td>-80.00 mV</td>
<td>-54.00 mV</td>
</tr>
<tr>
<td>$E_{\text{reset}}$</td>
<td></td>
<td>0.00 ms</td>
<td>2.00 mV</td>
</tr>
<tr>
<td>$\tau_{\text{ref}}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $\tau_e$ from 1 ms to 100 ms. The second sweep varies $E_{\text{Th}}$ from $-67.9 \text{ mV}$ 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 1048576 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, $n_g = 10$ and $n_g = 100$, referred to as $\text{ST}_{10}$ and $\text{ST}_{100}$ in the following.

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

$g_L = 0 \mu S$ and $\tau_e = 0$ 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 $E_{\text{Th}}$ is chosen as $E_L + \Delta_{\text{Th}} + 1 \text{ mV}$ to fulfill the constraints.
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 turn 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$_{10}$.

Table 4.9: Neuron parameter space exploration experiment runtimes for a $1024 \times 1024$ parameter sweep. The shown times correspond to the total CPU time, the wall-clock time is about $\frac{1}{2}$ of the given values. The results for the spike train measure with $n_g = 10$ are averaged over two runs.

<table>
<thead>
<tr>
<th></th>
<th>Sweep over $g_L$ and $\tau_e$</th>
<th>Sweep over $E_{Th}$ and $w$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIF</td>
<td>AdEx</td>
</tr>
<tr>
<td>ST$_{10}$</td>
<td>648 s</td>
<td>1595 s</td>
</tr>
<tr>
<td>ST$_{100}$</td>
<td>6578 s</td>
<td>13890 s</td>
</tr>
<tr>
<td>SGSO</td>
<td>206 s</td>
<td>310 s</td>
</tr>
<tr>
<td>SGMO</td>
<td>358 s</td>
<td>1014 s</td>
</tr>
</tbody>
</table>

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 $\tau_e$. 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.

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-
Figure 4.13: Selected results of the first design space exploration sweep (over $g_L$ and $\tau_e$). Colour-coded is the evaluation result $P \in [0, 1]$. 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 $E_{\text{Th}}$ is likely to produce many output spikes for small $E_{\text{Th}}$, 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.

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 $s^\text{in} = s^\text{out} = 3$ spikes, and the third a neuron population of $K = 3$ neurons.

For the LIF neuron, the only potential being optimised is $E_{\text{Th}}$, for the AdEx neuron $E_{\text{Th}}^\text{eff}$. Furthermore, $C_m$ is kept constant. Starting point of the optimisation are the initial neuron parameters $\Phi$ 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
4.7 Evaluation Method Comparison

<table>
<thead>
<tr>
<th>Scenarios for Neuron Parameter Optimisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant network parameters</td>
</tr>
<tr>
<td>SCENARIO I</td>
</tr>
<tr>
<td>Ones in the input c</td>
</tr>
<tr>
<td>Population size K</td>
</tr>
</tbody>
</table>

| Data encoding parameters                   |
| SCENARIO I | SCENARIO II | SCENARIO III |
| Input burst size s<sup>in</sup>             | 1           | 3           | 3           |
| Output burst size s<sup>out</sup>           | 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<sub>tt</sub>                       | 5 ms        | 5 ms        | 5 ms        |

| Resulting input and output spike counts     |
| SCENARIO I | SCENARIO II | SCENARIO III |
| Upper threshold n<sub>1</sub>                | 3           | 9           | 27          |
| Lower threshold n<sub>0</sub>                | 2           | 6           | 18          |
| Output spike count n<sub>out</sub>           | 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 <i>p</i><sub>1</sub>, <i>p</i><sub>0</sub>, <i>σ</i><sub>offs</sub><sup>t</sup>, <i>σ</i><sub>w</sub> are set to zero since they cannot be tested with the SGSO and SGMO measure, or, in the case of <i>σ</i><sub>offs</sub><sup>t</sup>, are not explicitly modelled. System parameters not listed here are irrelevant for single neuron evaluation.

The 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 <i>u</i> 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<sub>100</sub> 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<sub>100</sub> measure. Optimisation for SGMO results in mediocre values for both ST<sub>100</sub> 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

<table>
<thead>
<tr>
<th>Scenario</th>
<th>LIF</th>
<th>AdEx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ST$_{100}$</td>
<td>SGSO</td>
</tr>
<tr>
<td>Initial</td>
<td>0.78</td>
<td>0.43</td>
</tr>
<tr>
<td>II</td>
<td>0.28</td>
<td>/</td>
</tr>
<tr>
<td>III</td>
<td>0.00</td>
<td>/</td>
</tr>
</tbody>
</table>

### Final Evaluation Measure Optimality Values

<table>
<thead>
<tr>
<th>Target measure/Scenario</th>
<th>LIF</th>
<th>AdEx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ST$_{100}$</td>
<td>SGSO</td>
</tr>
<tr>
<td>I</td>
<td>1.00</td>
<td>0.40</td>
</tr>
<tr>
<td>II</td>
<td>0.52</td>
<td>/</td>
</tr>
<tr>
<td>III</td>
<td>0.00</td>
<td>/</td>
</tr>
<tr>
<td>ST$_{100}$</td>
<td>1.00</td>
<td>0.61</td>
</tr>
<tr>
<td>SGSO</td>
<td>0.73</td>
<td>0.33</td>
</tr>
<tr>
<td>II</td>
<td>0.42</td>
<td>/</td>
</tr>
<tr>
<td>III</td>
<td>0.44</td>
<td>/</td>
</tr>
</tbody>
</table>

Table 4.11: Initial and final neuron evaluation measure optimality values. The top part of the table shows the initial evaluation measure optimality values 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. $\diamond$ The final neuron parameter vectors for these results are shown in Table 4.12.

### Optimised LIF Neuron Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>[unit]</th>
<th>INIT.</th>
<th>ST$_{100}$</th>
<th>SGSO</th>
<th>SGMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Membrane cap.</td>
<td>$C_m$ [nF]</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>○ Leak conductance</td>
<td>$g_L$ [nS]</td>
<td>50.00</td>
<td>45.50</td>
<td>13.40</td>
<td>14.50</td>
</tr>
<tr>
<td>○ Leak potential</td>
<td>$E_L$ [mV]</td>
<td>-70.00</td>
<td>-70.00</td>
<td>-70.00</td>
<td>-70.00</td>
</tr>
<tr>
<td>○ Threshold potential</td>
<td>$E_{Th}$ [mV]</td>
<td>-54.00</td>
<td>-52.50</td>
<td>-48.23</td>
<td>-65.80</td>
</tr>
<tr>
<td>○ Reset potential</td>
<td>$E_{reset}$ [mV]</td>
<td>-80.00</td>
<td>-80.00</td>
<td>-80.00</td>
<td>-80.00</td>
</tr>
<tr>
<td>○ Refractory period</td>
<td>$\tau_{ref}$ [ms]</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>○ Synapse potential</td>
<td>$E_e$ [mV]</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>○ Synapse time</td>
<td>$\tau_e$ [ms]</td>
<td>5.00</td>
<td>5.37</td>
<td>0.10</td>
<td>1.20</td>
</tr>
<tr>
<td>○ Synapse weight</td>
<td>$w$ [nS]</td>
<td>30.00</td>
<td>36.25</td>
<td>1477</td>
<td>19.23</td>
</tr>
</tbody>
</table>

Table 4.12: Initial and optimised LIF neuron parameters. The corresponding optimality values $P$ are shown in Table 4.11, where they are marked with a $\diamond$-symbol. $\circ$ These parameter dimensions were optimised.
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 $\tau_e$, 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 $P_{st} = 0.5$, 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 ($P_{st} < 0.5$). As long as the neuron should only produce a single output spike, the SGSO measure seems to be a viable option for parameter selection.

**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
neuron parameter evaluation and 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.
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 $\Phi$ (Section 4.1.1), a test dataset $D$ 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
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.

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).

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.

A short usage example of PyNNLess is provided in Appendix A.2.
5.1 Methodology and Software Architecture

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 $D$ 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 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.

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.

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.

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 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.

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.

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

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

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.

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 too 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 $E_e = 0$ mV and $C_m = 0.2$ 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 $-55$ mV. The parameters $g_L$ and $\tau_{ref}$ can be set individually per neuron. The synapse time constants $\tau_e$ and $\tau_i$ 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 discuss 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 \( \mathcal{D} \) generated by PyNAM, achieves a maximum storage capacity of \( I = 227 \) bit with ten expected false positives over all test samples.

The neuron parameter vectors are sampled from a uniformly spaced \( 64 \times 64 \) grid. For each parameter vector, the false positive and negative counts \( n_{fp}, n_{fn} \), the information measure \( I \), and the average latency \( \delta \) 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 \( n_{fp} = -1 \) corresponds to “none of the expected false positives are generated”, \( n_{fp} = 0 \) to “the expected number of false positives was generated” and \( n_{fp} = 1 \) 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.

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 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 \([0, 1]\).
Figure 5.2: Comparison of the $g_L/\tau_e$ 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 first parameter sweep executed on NM-MC1 varies $g_L$ from $0.01 \mu S$ to $0.2 \mu S$ and $\tau_e$ from $0 \text{ ms}$ to $20 \text{ ms}$. 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 $0.01 \mu S$ to $0.2 \mu S$ and $\tau_e$ from $0 \text{ ms}$ to $20 \text{ ms}$. As shown in Figure 5.3(a), the region of
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 $E_{\text{Th}}/w$ sweep on NEST and NM-MC1. Refer to Sections 5.2.1 and 5.2.2 for more information.
5.2 Neuron parameter evaluation

5.2.3 Neuron parameter sweep on Spikey

The excitatory channel time constant $\tau_e$ is not user-definable on Spikey. Correspondingly, the $g_L/\tau_e$ sweep cannot be performed. The threshold potential $E_{Th}$ is furthermore limited to a maximum of $-55$ mV, which lies outside the range of the previous sweep. A maximally large range (with respect to $E_l$ at $-70$ mV) is chosen instead and $E_{Th}$ is varied from $-69$ mV to $-55$ mV. The synaptic channel amplitude is reduced along with $E_{Th}$ by lowering the $w$-range to $0.0 \mu$S to $0.016 \mu$S. Note that the membrane capacitance of Spikey is $C_m = 0.2$ nF instead of the previously used $1.0$ 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$^{100}_1$ 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,
Figure 5.5: Comparison of the $E_{\text{Th}}/w$ 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.
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.

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.
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 $\sigma$, 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 $\Phi$ 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 $\Phi$ 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 $C_m = 0.2 \text{nF}$ and $E_{Th} = -55 \text{mV}$. According to the neuron parameter sweep in Figure 5.5(a), a maximum in the storage capacity is reached for $w = 16 \text{nS}$. A preliminary test run on NEST shows that these parameters reach the theoretical maximum storage capacity if the spike time jitter is reduced from $\sigma_t = 5 \text{ms}$ to $2 \text{ms}$ in the initial system parameters.

In the following, the system parameters $\sigma_t$, $\sigma_w$ and $T$ are linearly sampled from a given value range in fifty steps. For each sample, the
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 $\sigma_t$ is varied from 0 ms to 20 ms. The results are shown in Figure 5.7. As expected, the information measure decreases with increasing $\sigma_t$. 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 $\sigma_t = 0$ ms for ESS is caused by a maximally large initial false negative count $n_{fn} = 81$. 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 $\sigma_w$ (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 $\sigma_t$, 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.
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.
CONCLUSION AND OUTLOOK

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.

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.

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].
CONCLUSION AND OUTLOOK

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 Human Brain Project, and a yet so small step towards the development of artificial cognitive systems.
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:

```c++
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:

```c++
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.

```python
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"][:,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 $\tau_e$.

```json
{
    "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 g_L, 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
        }
    ]
}
```
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.

<table>
<thead>
<tr>
<th>j</th>
<th>( a_{\ell j} )</th>
<th>( b_j )</th>
<th>( a_{\ell j} )</th>
<th>( b_j )</th>
<th>( a_{\ell j} )</th>
<th>( b_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1/6</td>
<td>1/3</td>
<td>1/3</td>
</tr>
<tr>
<td>( 1/2 )</td>
<td>1/2</td>
<td>( \frac{1}{2} )</td>
<td>( \frac{1}{2} )</td>
<td>( \frac{1}{2} )</td>
<td>( \frac{1}{2} )</td>
<td>( \frac{1}{2} )</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table B.2:** Runge-Kutta coefficients for the fifth-order Dormand-Prince integrator, which can be used as adaptive step size integrator [DP80].

<table>
<thead>
<tr>
<th>( b_j )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{35}{56} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \frac{19372}{6561} )</td>
<td>- ( \frac{5247}{6561} )</td>
<td>( \frac{5103}{6561} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( a_{\ell j} )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>( \frac{5}{6} )</td>
<td>( \frac{5}{6} )</td>
<td>( \frac{5}{6} )</td>
<td>( \frac{5}{6} )</td>
<td>( \frac{5}{6} )</td>
<td>( \frac{5}{6} )</td>
<td>( \frac{5}{6} )</td>
</tr>
<tr>
<td>3</td>
<td>( \frac{3}{2} )</td>
<td>( \frac{3}{2} )</td>
<td>( \frac{3}{2} )</td>
<td>( \frac{3}{2} )</td>
<td>( \frac{3}{2} )</td>
<td>( \frac{3}{2} )</td>
<td>( \frac{3}{2} )</td>
</tr>
<tr>
<td>4</td>
<td>( \frac{44}{15} )</td>
<td>( \frac{85}{15} )</td>
<td>( \frac{85}{15} )</td>
<td>( \frac{85}{15} )</td>
<td>( \frac{85}{15} )</td>
<td>( \frac{85}{15} )</td>
<td>( \frac{85}{15} )</td>
</tr>
<tr>
<td>5</td>
<td>( \frac{59}{15} )</td>
<td>( \frac{59}{15} )</td>
<td>( \frac{59}{15} )</td>
<td>( \frac{59}{15} )</td>
<td>( \frac{59}{15} )</td>
<td>( \frac{59}{15} )</td>
<td>( \frac{59}{15} )</td>
</tr>
<tr>
<td>6</td>
<td>( \frac{9017}{180} )</td>
<td>( \frac{9017}{180} )</td>
<td>( \frac{9017}{180} )</td>
<td>( \frac{9017}{180} )</td>
<td>( \frac{9017}{180} )</td>
<td>( \frac{9017}{180} )</td>
<td>( \frac{9017}{180} )</td>
</tr>
<tr>
<td>7</td>
<td>( \frac{384}{1} )</td>
<td>( \frac{384}{1} )</td>
<td>( \frac{384}{1} )</td>
<td>( \frac{384}{1} )</td>
<td>( \frac{384}{1} )</td>
<td>( \frac{384}{1} )</td>
<td>( \frac{384}{1} )</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Time</th>
<th>Module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.47%</td>
<td>libm-2.22</td>
<td>__ieee754_exp_sse2</td>
</tr>
<tr>
<td>15.24%</td>
<td>◦</td>
<td>BenchmarkResult::compare</td>
</tr>
<tr>
<td>13.97%</td>
<td>◦</td>
<td>Model::simulate&lt;DormandPrinceIntegrator&gt;</td>
</tr>
<tr>
<td>11.05%</td>
<td>libc-2.22</td>
<td>__memmove_ssse3</td>
</tr>
<tr>
<td>6.56%</td>
<td>◦</td>
<td>RungeKuttaIntegrator::integrate</td>
</tr>
<tr>
<td>4.18%</td>
<td>◦</td>
<td>benchmarkSimple&lt;MidpointIntegrator&gt;</td>
</tr>
<tr>
<td>3.97%</td>
<td>◦</td>
<td>std::vector::emplace_back&lt;double&gt;</td>
</tr>
<tr>
<td>3.23%</td>
<td>◦</td>
<td>benchmarkSimple&lt;EulerIntegrator&gt;</td>
</tr>
<tr>
<td>2.51%</td>
<td>◦</td>
<td>Model::simulate&lt;RungeKuttaIntegrator&gt;</td>
</tr>
</tbody>
</table>

**Table B.4:** Single neuron simulator runtime profile with approximated exponential.

<table>
<thead>
<tr>
<th>Time</th>
<th>Module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.06%</td>
<td>◦</td>
<td>Model::simulate&lt;DormandPrinceIntegrator&gt;</td>
</tr>
<tr>
<td>17.18%</td>
<td>◦</td>
<td>BenchmarkResult::compare</td>
</tr>
<tr>
<td>14.05%</td>
<td>libc-2.22</td>
<td>__memmove_ssse3</td>
</tr>
<tr>
<td>7.30%</td>
<td>◦</td>
<td>benchmarkSimple&lt;MidpointIntegrator&gt;</td>
</tr>
<tr>
<td>7.21%</td>
<td>◦</td>
<td>RungeKuttaIntegrator::integrate</td>
</tr>
<tr>
<td>5.26%</td>
<td>◦</td>
<td>benchmarkSimple&lt;EulerIntegrator&gt;</td>
</tr>
<tr>
<td>5.01%</td>
<td>◦</td>
<td>Model::aux</td>
</tr>
<tr>
<td>4.54%</td>
<td>◦</td>
<td>std::vector::emplace_back&lt;double&gt;</td>
</tr>
<tr>
<td>3.41%</td>
<td>◦</td>
<td>Model::simulate&lt;RungeKuttaIntegrator&gt;</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Integrator</th>
<th>Time and samples</th>
<th>Error (RMSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t$ [ms]</td>
<td>$N$</td>
</tr>
<tr>
<td>Euler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h = 1$ µs</td>
<td>2476.087</td>
<td>10011577</td>
</tr>
<tr>
<td>$h = 10$ µs</td>
<td>209.023</td>
<td>1002278</td>
</tr>
<tr>
<td>$h = 100$ µs</td>
<td>18.174</td>
<td>101255</td>
</tr>
<tr>
<td>$h = 1$ ms</td>
<td>2.034</td>
<td>11190</td>
</tr>
<tr>
<td>Midpoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h = 1$ µs</td>
<td>3609.626</td>
<td>10011577</td>
</tr>
<tr>
<td>$h = 10$ µs</td>
<td>329.711</td>
<td>1002278</td>
</tr>
<tr>
<td>$h = 100$ µs</td>
<td>29.946</td>
<td>101249</td>
</tr>
<tr>
<td>$h = 1$ ms</td>
<td>3.047</td>
<td>11161</td>
</tr>
<tr>
<td>Runge-Kutta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h = 1$ µs</td>
<td>5801.178</td>
<td>10011577</td>
</tr>
<tr>
<td>$h = 10$ µs</td>
<td>542.762</td>
<td>1002278</td>
</tr>
<tr>
<td>$h = 100$ µs</td>
<td>56.256</td>
<td>101250</td>
</tr>
<tr>
<td>$h = 1$ ms</td>
<td>5.410</td>
<td>11149</td>
</tr>
<tr>
<td>Dormand-Prince</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e = 1$ µ</td>
<td>3244.054</td>
<td>1934005</td>
</tr>
<tr>
<td>$e = 10$ µ</td>
<td>774.668</td>
<td>699606</td>
</tr>
<tr>
<td>$e = 100$ µ</td>
<td>80.463</td>
<td>74549</td>
</tr>
<tr>
<td>$e = 1$ m</td>
<td>11.903</td>
<td>10783</td>
</tr>
<tr>
<td>$e = 10$ m</td>
<td>5.444</td>
<td>4549</td>
</tr>
<tr>
<td>$e = 100$ m</td>
<td>4.791</td>
<td>3934</td>
</tr>
</tbody>
</table>
### Table B.6: Differential equation integrator performance for the IfCondExp model. Errors are RMSE values compared to a Runge-Kutta integration with 100 nssample time. Percentages represent RMSE normalised to the value range. Simulated time is 10 sat 100 input spike groups with 33 output spikes.

<table>
<thead>
<tr>
<th>Integrator</th>
<th>( h = 1 ) µs</th>
<th>( h = 10 ) µs</th>
<th>( h = 100 ) µs</th>
<th>( h = 1 ) ms</th>
<th>( h = 10 ) ms</th>
<th>( h = 100 ) ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euler</td>
<td>2071.354</td>
<td>130.804</td>
<td>12.139</td>
<td>1.198</td>
<td>1.482</td>
<td>3.024</td>
</tr>
<tr>
<td>Midpoint</td>
<td>2412.932</td>
<td>165.374</td>
<td>13.282</td>
<td>1.482</td>
<td>3.024</td>
<td>7.367</td>
</tr>
<tr>
<td>Runge-Kutta</td>
<td>3873.598</td>
<td>314.017</td>
<td>32.832</td>
<td>3.024</td>
<td>7.367</td>
<td>2.457</td>
</tr>
<tr>
<td>Dormand-Prince</td>
<td>2286.730</td>
<td>314.017</td>
<td>32.832</td>
<td>3.024</td>
<td>7.367</td>
<td>2.457</td>
</tr>
</tbody>
</table>

**Error (RMSE)**

- Time and samples
- Input: \( \Delta t = 0.001 \) ms
- Output: \( \Delta t = 0.001 \) ms
- Percentage normalised to the value range

**Table B.6**: Different error integrator performance for the IfCondExp model. Errors are RMSE values compared to a Runge-Kutta integration with 100 nssample time. Percentages represent RMSE normalised to the value range. Simulated time is 10 sat 100 input spike groups with 33 output spikes.
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.

<table>
<thead>
<tr>
<th>Integrator</th>
<th>Time and samples</th>
<th>Error (RMSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t$ [ms]</td>
<td>$N$</td>
</tr>
<tr>
<td>Euler</td>
<td>$h = 1$ µs</td>
<td>2244.048</td>
</tr>
<tr>
<td></td>
<td>$h = 10$ µs</td>
<td>182.803</td>
</tr>
<tr>
<td></td>
<td>$h = 100$ µs</td>
<td>15.416</td>
</tr>
<tr>
<td></td>
<td>$h = 1$ ms</td>
<td>1.651</td>
</tr>
<tr>
<td>Midpoint</td>
<td>$h = 1$ µs</td>
<td>2994.206</td>
</tr>
<tr>
<td></td>
<td>$h = 10$ µs</td>
<td>264.261</td>
</tr>
<tr>
<td></td>
<td>$h = 100$ µs</td>
<td>23.706</td>
</tr>
<tr>
<td></td>
<td>$h = 1$ ms</td>
<td>2.579</td>
</tr>
<tr>
<td>Runge-Kutta</td>
<td>$h = 1$ µs</td>
<td>5045.481</td>
</tr>
<tr>
<td></td>
<td>$h = 10$ µs</td>
<td>460.079</td>
</tr>
<tr>
<td></td>
<td>$h = 100$ µs</td>
<td>47.700</td>
</tr>
<tr>
<td></td>
<td>$h = 1$ ms</td>
<td>4.560</td>
</tr>
<tr>
<td>Dormand-Prince</td>
<td>$e = 1$ µ</td>
<td>2989.163</td>
</tr>
<tr>
<td></td>
<td>$e = 10$ µ</td>
<td>735.503</td>
</tr>
<tr>
<td></td>
<td>$e = 100$ µ</td>
<td>74.478</td>
</tr>
<tr>
<td></td>
<td>$e = 1$ m</td>
<td>10.839</td>
</tr>
<tr>
<td></td>
<td>$e = 10$ m</td>
<td>4.739</td>
</tr>
<tr>
<td></td>
<td>$e = 100$ m</td>
<td>4.205</td>
</tr>
</tbody>
</table>
SINGLE NEURON EVALUATION COMPARISON

SGSO evaluation model (AdEx neuron)

Leak conductivity $g_L$ [µS]

Excitatory time constant $\tau_e$ [ms]

Threshold potential $E_{Th}$ [mV]

Synapse weight $w$ [µS]

Evaluation result $P_{sgso}(\Phi)$
SINGLE NEURON EVALUATION COMPARISON

SGMO evaluation model (AdEx neuron)

1. Leak conductivity $g_L$ [µS]
   - 20
   - 40
   - 60
   - 80
   - 100

2. Excitatory time constant $\tau_e$ [ms]
   - 0.0
   - 0.1
   - 0.2
   - 0.3
   - 0.4
   - 0.5
   - 0.6

3. Threshold potential $E_{Th}$ [mV]
   - 0.0
   - 0.2
   - 0.4
   - 0.6
   - 0.8
   - 1.0

4. Synapse weight $w$ [µS]
   - 0.0
   - 0.1
   - 0.2
   - 0.3
   - 0.4
   - 0.5
   - 0.6
   - 0.7
   - 0.8
   - 0.9
   - 1.0

Evaluation result $P_{sgmo}(\Phi)$
Spike Train evaluation model with $n_g = 100$ (AdEx neuron)

Leak conductivity $g_L$ [µS]
20
40
60
80
100

Excitatory time constant $\tau_e$ [ms]

Spike Train evaluation model with $n_g = 100$ (AdEx neuron)

Threshold potential $E_{Th}$ [mV]
0.0
0.2
0.4
0.6
0.8
1.0

Synapse weight $w$ [µS]
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

Evaluation result $P_{st}(\Phi)$
SINGLE NEURON EVALUATION COMPARISON

SGMO evaluation model (LIF neuron)

Leak conductivity $g_L$ [µS]
20 40 60 80 100

Excitatory time constant $\tau_e$ [ms]

Evaluation result $P_{sgmo}(\Phi)$

Threshold potential $E_{Th}$ [mV]

Synapse weight $w$ [µS]

Evaluation result $P_{sgmo}(\Phi)$
Spike Train evaluation model with $n_g = 100$ (LIF neuron)

Leak conductivity $g_L$ [$\mu$S]

| 20 | 40 | 60 | 80 | 100 |

Excitatory time constant $\tau_e$ [ms]

Spike Train evaluation model with $n_g = 100$ (LIF neuron)

Evaluation result $P_{st}(\Phi)$

Threshold potential $E_{Th}$ [mV]

Synapse weight $w$ [$\mu$S]

Spike Train evaluation model with $n_g = 100$ (LIF neuron)

Evaluation result $P_{st}(\Phi)$

Threshold potential $E_{Th}$ [mV]
Single Neuron Evaluation Comparison

**SGSO Evaluation Model (LIF Neuron)**

- **Leak Conductivity** $g_L$ [$\mu$S]
  - Values: 20, 40, 60, 80, 100

- **Excitatory Time Constant** $\tau_e$ [ms]
  - Values: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

- **Threshold Potential** $E_{\text{Th}}$ [mV]
  - Values: 0.0

- **Synapse Weight** $w$ [$\mu$S]
  - Values: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

- **Evaluation Result** $P_{SGSO}(\Phi)$
  - Values: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

---

**Graphs**

1. **Leak Conductivity vs. Excitatory Time Constant**
2. **Threshold Potential vs. Synapse Weight**
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdEx</td>
<td>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</td>
</tr>
<tr>
<td>AdExpSim</td>
<td>Adaptive exponential neuron simulator framework. Collection of libraries and applications for single neuron evaluation. 90, 92, 93, 99, 119, 121, 123</td>
</tr>
<tr>
<td>API</td>
<td>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</td>
</tr>
<tr>
<td>BiNAM</td>
<td>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</td>
</tr>
<tr>
<td>BrainScaleS</td>
<td>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</td>
</tr>
<tr>
<td>CLI</td>
<td>Command line interface. 90, 92</td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit. 96, 105, 126</td>
</tr>
<tr>
<td>DoF</td>
<td>Degree of freedom. 68, 69</td>
</tr>
<tr>
<td>EIF</td>
<td>Exponential integrate-and-fire neuron model. 20, 21</td>
</tr>
<tr>
<td>ESS</td>
<td>Executable system specification. A software emulator of the NM-PM1 system. 26, 27, 104, 106, 114–117</td>
</tr>
<tr>
<td>FACETS</td>
<td>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</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics processing unit. Massively parallel processor system primarily designed for 3D graphics processing. In the recent years also usable for general purpose computing. 121</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface. 90, 92</td>
</tr>
<tr>
<td>HBP</td>
<td>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</td>
</tr>
</tbody>
</table>
ACRONYMS

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 $\Delta t$. 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:

\[ E = \sqrt{\frac{1}{N} \sum_{k=1}^{N} ||\vec{t}_k - \vec{x}_k||^2}, \]

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

\(\alpha\) Dimensionality of the neuron state vector \(\vec{v} \in \mathbb{R}^\alpha\). 73

\(B\) Base set of Boolean algebra, defined as \(B = \{0, 1\}\). 6, 9, 30–32, 54, 55, 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 \(c = \sum_i^n (\vec{x})_i\). 30–34, 36, 46, 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

\(D\) Test dataset. Consists of \(N\) input and output vectors \(D = \{\vec{x}_k, \vec{y}_k\}\). 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

\(\delta\) Latency, time until the entire output for an input has been produced. 51, 65, 107, 110, 113

\(\Delta_a\) Adaptation induced voltage change, substitute of the state variable \(l_a(t)\). 69, 71, 73

\(\Delta_b\) Spike induced adaptation, substitute of the parameter \(b\). 69, 71

\(\Delta t\) Interspike interval (ISI), the equidistant delay between two spikes in a single spike burst in seconds. 22, 44–48, 64, 78, 81, 99

\(\Delta T_h\) Spike slope factor in the AdEx model. 21–23, 64, 67–69, 72, 83, 95

\(\Delta u\) 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
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

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

Spike potential, the maximum potential reached during a spike. 14, 19, 21, 22, 68

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

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

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

Subthreshold adaptation, substitute of the parameter $a$. 69

Current excitatory channel frequency, substitute of the state variable $g_e(t)$. 69, 71, 73

Current inhibitory channel frequency, substitute of the state variable $g_i(t)$. 69, 71, 73

Membrane leak channel rate, substitute of the parameter $g_L$. 69

Synaptic weight, substitute of the parameter $w$. 69

Excitatory synapse conductance in siemens. 19, 23, 66, 69, 127–129

Inhibitory synapse conductance in siemens. 19, 23, 69, 83, 127–129


Heaviside function $H(x)$, named after the mathematician and physicist Oliver Heaviside. 9, 10, 31

Differential equation integrator step size. 71–76, 113

Information (or entropy) stored in a memory in bits. 32–34, 37, 107–109, 111, 112, 114–117

Place holder for an ion species with either positive or negative charge. 12, 13

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

Adaptation current in the AdEx neuron model. 21, 23, 69, 72, 83, 90

Neuron model specific intrinsic component of the neuronal current $i$. 17, 18
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{\text{syn}}$</td>
<td>Synaptic or external component of the neuronal current $i$. 16, 17, 19–21</td>
</tr>
<tr>
<td>$I^k_{\text{syn}}$</td>
<td>Current induced by a single synapse $k$. 17, 18</td>
</tr>
<tr>
<td>$I_{\text{Th}}$</td>
<td>Exponential threshold current used in the AdEx model. 21, 22, 72, 80, 83</td>
</tr>
<tr>
<td>$I_{\text{max}}_{\text{Th}}$</td>
<td>Maximum the exponential threshold current $I_{\text{Th}}$ is limited to for numerical integration. 72</td>
</tr>
<tr>
<td>$K$</td>
<td>Population size. Number of neurons in the network output layer representing a single output component. 39, 46–48, 64, 78, 98, 99</td>
</tr>
<tr>
<td>$k^{\text{in}}$</td>
<td>List of sample indices. Assigns a sample number to each input spike time in $t^{\text{in}}$. 47, 103</td>
</tr>
<tr>
<td>$\lambda_a$</td>
<td>Adaptation channel decay rate, substitute of the parameter $\tau_a$. 69</td>
</tr>
<tr>
<td>$\lambda_e$</td>
<td>Excitatory channel decay rate, substitute of the parameter $\tau_e$. 69</td>
</tr>
<tr>
<td>$\lambda_i$</td>
<td>Inhibitory channel decay rate, substitute of the parameter $\tau_i$. 69</td>
</tr>
<tr>
<td>$M$</td>
<td>Binary matrix of size $m \times n$, storing the trained associations of the network. 30–32, 34–37, 39, 40, 53–55, 103</td>
</tr>
<tr>
<td>$m$</td>
<td>Input dimensionality of the BiNAM. 30–36, 42, 46, 47, 52–55, 64, 67, 107</td>
</tr>
<tr>
<td>$\mu^\text{offs}_t$</td>
<td>Random offset chosen for an entire spike burst. This value is once sampled for a spike burst from a Gaussian distribution with standard deviation $\sigma^\text{offs}_t$. 45</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of samples trained in the BiNAM. 30, 33, 34, 36, 46, 47, 51–61, 64, 107, 141</td>
</tr>
<tr>
<td>$n$</td>
<td>Output dimensionality of the BiNAM. 30, 32–36, 42, 46, 52–54, 64, 66, 67, 107</td>
</tr>
<tr>
<td>$n_0$</td>
<td>Maximum number of input spikes for which a neuron in the spiking BiNAM implementation should produce no output spikes. 48, 79, 80, 95, 99</td>
</tr>
<tr>
<td>$n_1$</td>
<td>Minimum number of input spikes for which a neuron in the spiking BiNAM implementation should produce $s^{\text{out}}$ output spikes. 48, 79, 80, 95, 99</td>
</tr>
<tr>
<td>$n_E$</td>
<td>Number of excitatory input bursts in an experiment group descriptor of the “spike train” evaluation measure. 77, 78</td>
</tr>
<tr>
<td>$n_I$</td>
<td>Number of inhibitory input bursts in an experiment group descriptor of the “spike train” evaluation measure. 77, 78</td>
</tr>
<tr>
<td>$n_{\text{fn}}$</td>
<td>Total number of false negative bits across the entire test dataset. 107, 110, 116</td>
</tr>
<tr>
<td>$n^{k}_{\text{fn}}$</td>
<td>Number of false negative bits for the recall of the $k$-th sample. 33, 49</td>
</tr>
<tr>
<td>$n_{\text{fp}}$</td>
<td>Total number of false positive bits across the entire test dataset. 107, 110–112</td>
</tr>
<tr>
<td>$n^{k}_{\text{fp}}$</td>
<td>Number of false positive bits for the recall of the $k$-th sample. 33, 34, 49</td>
</tr>
</tbody>
</table>
SYMBOLS

\( n_g \) 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

\( n_{out}^i \) Actual number of received output spikes in the \( i \)-th experiment group. 78, 79

\( \bar{n}_{out} \) Expected number of output spike bursts in an experiment group descriptor of the “spike train” evaluation measure. 77, 78, 84, 85, 87, 94, 99

\( \bar{n}_{out}^i \) 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

\( \Phi \) Abstract spiking neuron parameter vector. 62–66, 76–79, 82, 84–86, 89, 94, 98, 103, 114

\( \phi \) 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 spikes and 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

\( \sigma_\phi \) Standard deviation of a single neuron parameter \( \phi \). 64

\( \sigma_t \) 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

\( \sigma_{offs} \) Standard deviation of the random offset \( \mu_{offs} \) for an entire burst. 45–47, 50, 64, 78, 80, 81, 99
**Symbols**

- $\sigma_w$: Standard deviation of the Gaussian noise added to the synaptic weight $w$, special case of $\sigma_g$. 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
- $\tau_a$: Adaptation current time constant, controls the exponential decay of the adaptation current $I_a$ (in seconds). 21, 23, 64, 67–69, 95
- $\tau_e$: 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
- $\tau_i$: Inhibitory synapse time constant, controls the exponential decay of inhibitory channel conductance $g_i$. 18, 19, 23, 64, 67–69, 106
- $\tau_m$: Membrane potential time constant, controls the exponential decay of the membrane potential if no external input current is present. It holds $\tau_m = C_m/g_L$. Measured in seconds. 23
- $\tau_{ref}$: 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
- $\Theta$: Vectorial threshold function. $\Theta_{\theta}(\vec{z})$ returns a copy of $\vec{z}$ with all components greater of equal to $\theta$ set to one and all other values set to zero. 31, 34
- $\theta$: 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
- $t^{in}_i$: 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
- $t^{out}_j$: 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
- $\dot{u}$: 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
**SYMBOLS**

\( \vec{v} \) 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

\( w_{\text{in}} \) 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

\( \hat{y} \) \( n \)-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


ERKLÄRUNG


Bielefeld, den 21. Dezember 2015  

__________________________  
(Unterschrift)