DEEP SPIKING NEURAL NETWORKS

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

2018

Qian Liu
School of Computer Science

Contents

Word count 38896

List of Tables

List of Figures

The University of Manchester

Qian Liu
Doctor of Philosophy
Deep Spiking Neural Networks
February 5, 2018

Neuromorphic Engineering (NE) has led to the development of biologically-inspired computer architectures whose long-term goal is to approach the performance of the human brain in terms of energy efficiency and cognitive capabilities. Although there are a number of neuromorphic platforms available for large-scale Spiking Neural Network (SNN) simulations, the problem of programming these brain-like machines to be competent in cognitive applications still remains unsolved. On the other hand, Deep Learning has emerged in Artificial Neural Network (ANN) research to dominate state-of-the-art solutions for cognitive tasks. Thus the main research problem emerges of understanding how to operate and train biologically-plausible SNNs to close the gap in cognitive capabilities between SNNs and ANNs.

SNNs can be trained by first training an equivalent ANN and then transferring the tuned weights to the SNN. This method is called 'off-line' training, since it does not take place on an SNN directly, but rather on an ANN instead. However, previous work on such off-line training methods has struggled in terms of poor modelling accuracy of the spiking neurons and high computational complexity. In this thesis we propose a simple and novel activation function, Noisy Softplus (NSP), to closely model the response firing activity of biologically-plausible spiking neurons, and introduce a generalised off-line training method using the Parametric Activation Function (PAF) to map the abstract numerical values of the ANN to concrete physical units, such as current and firing rate in the SNN. Based on this generalised training method and its fine tuning, we achieve the state-of-the-art accuracy on the MNIST classification task using spiking neurons, 99.07%, on a deep spiking convolutional neural network (ConvNet).

We then take a step forward to 'on-line' training methods, where Deep Learning modules are trained purely on SNNs in an event-driven manner. Existing work has failed to provide SNNs with recognition accuracy equivalent to ANNs due to the lack of mathematical analysis. Thus we propose a formalised Spike-based Rate Multiplication (SRM) method which transforms the product of firing rates to the number of coincident spikes of a pair of rate-coded spike trains. Moreover, these coincident spikes can be captured by the Spike-Time-Dependent Plasticity (STDP) rule to update the weights between the neurons in an on-line, event-based, and biologically-plausible manner. Furthermore, we put forward solutions to reduce correlations between spike trains; thereby addressing the result of performance drop in on-line SNN training. The promising results of spiking Autoencoders (AEs) and Restricted Boltzmann Machines (SRBMs) exhibit equivalent, sometimes even superior, classification and reconstruction capabilities compared to their non-spiking counterparts.

To provide meaningful comparisons between these proposed SNN models and other existing methods within this rapidly advancing field of NE, we propose a large dataset of spike-based visual stimuli and a corresponding evaluation methodology to estimate the overall performance of SNN models and their hardware implementations.

Declaration

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

Copyright Statement

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University's Policy on Presentation of Theses.

Acknowledgements

First and foremost I would like to express my deepest appreciation to my supervisor, Prof. Steve Furber, for all his contributions of time, patience, ideas and funding to my study. I am grateful for his 'ease-the-grip' strategy that I have complete freedom to explore my own research problems and find out solutions to them. Although sometimes problems occur and tough time comes, he tells and encourages me, persistence is the key to all the problems. It builds up my confidence to think and act professionally like a scientist. At the same time, he knows when to 'hold hands firmly' that the thesis would not be finished without his kind reminders about commitment on writing.

I would like to thank my colleagues John Woods, Cameron Patterson, Garibaldi Pineda Garcia, Yunhua Chen, Jonathan Heathcote and Robert James, for their collaborations, meaningful discussions, insightful comments and helpful English revisions on my papers and thesis. Those sleepless nights we work towards deadlines, and the colourful and densely packed corrections on my printed papers will stay in my memory.

In addition, I express my gratitude to Cameron Patterson, Evangelos Stromatias, Mireya Lopez, Guangda Zhang, Garibaldi Garcia, Gengting Liu, Athanasios Stratikopoulos, and Crefeda Rodrigues for being my great companions and friends. My study would not be such an enjoyable experience without them, since they always celebrate any achievement I have made, pull me out of stress and negative moods towards the bright side, and share the bitter-sweets on the journey to our PhD.

My sincere thanks to my parents, Caixia Zhou and Yancheng Liu, beloved husband, Xuekai Li and long-time friends Jingjing Cui and Xiaowen Jia for supporting and understanding me spiritually throughout my life in general. I cannot be who I am without being part of their lives. Especially with my husband, we take the adventure to Manchester, learn about the world, understand ourselves and pursue the meaning of life together. My special thanks to him again.

List of Acronyms

AE Autoencoder

AER Address-Event Representation

AI Artificial Intelligence

ANN Artificial Neural Network

BP Backpropagation

CA Classification Accuracy

CD Contrastive Divergence

Conv convolutional

ConvNet Convolutional Networks

DBN Deep Belief Network

DVS Dynamic Visual Sensor

EPSP Excitatory Post-Synaptic Potential

FC fully-connected

FoCal Filter Overlap Correction ALgorithm

GAN Generative Adversarial Network

IHC Inner Hair Cell

IPSP Inhibitory Post-Synaptic Potential

LIF Leaky Integrate-and-Fire

LSTM Long Short-Term Memory

MLP Multilayer Perceptron

MSE Mean Squared Error

NE Neuromorphic Engineering

NI noisy input

NSP Noisy Softplus

PAF Parametric Activation Function

Pool pooling

PSP Post-Synaptic Potential

RC Resistor-Capacitor

ReLU Rectified Linear Unit

RBM Restricted Boltzmann Machine

RNN Recurrent Neural Network

ROC Rank Order Coding

SAE Spiking Autoencoder

SGC Spiral Ganglion Cell

SGD Stochastic Gradient Descent

SNN Spiking Neural Network

Sopbs/W synaptic operations per biological second per Watt

Sops/W synaptic operations per second per Watt

SRBM Spiking Restricted Boltzmann Machine

SRM Spike-based Rate Multiplication

STDP Spike-Time-Dependent Plasticity

VLSI Very-Large-Scale Integration

WTA Winner-Take-All

List of Symbols

k	data index or the noise scaling factor of Noisy Softplus
l	log likelihood
δ	error gradient
η	learning rate
λ	firing rate
μ_v	instantaneous mean of the change in membrane potential
σ_v	standard deviation of the change in membrane potential
σ	sigmoid function or the second parameter, noise level, of Noisy Softplus
τ_{dur}	duration of spike train
τ_{refrac}	refractory period
τ_{win}	window length in time
τ_{syn}	synaptic time constant
τ_m	membrane time constant
Θ	parameter
θ	parameter

b	bias
C_m	membrane capacitance
D	dataset
dt	time step
E_{syn}	reversal potential of the ion channel in a synapse
E	error or energy
f	function
g_{syn}	conductance of the ion channel in a synapse
h	value of a hidden unit
I_{offset}	offset of input current
I_c	current which charges the capacitor
I_r	current which discharges through the resistance
I	current
K	number of data samples or scaling factor maps numerical numbers to firing rates in Hz
L	loss or likelihood
m_I	mean of the input current
N	total number
net	weighted sum, the input value of an activation function
p	possibility or parameter p of Parametric Activation Function
R_m	membrane resistance

s_I	standard deviation of the input current
S	mapping scaling factor
s	spike train
t	time or teaching labels
T	total time
V_{reset}	resetting potential after a neuron spikes
V_{rest}	resting potential
V_{th}	threshold of the membrane potential
V	membrane potential
v	value of a visible unit
W_t	Wiener process
w	weight or synaptic strength
x	input of a model or a function
y	output of a model or a function
Z	partition function

Chapter 1

Introduction

"As engineers, we would be foolish to ignore the lessons of a billion years of evolution."

- Carver Mead, *1993*

Advances in computing power and machine learning have endowed computers with rapidly growing performance in cognitive tasks such as recognising objects [Deng et al., 2009] and playing GO [Silver et al., 2016]; these tasks were once dominated by human intelligence and solved by biological neurons in the brain. However, humans and many other animals still outperform computers in practical tasks, such as vision, and in terms of size and energy cost by several orders of magnitude. For instance, AlphaGO [Silver et al., 2016] has a power consumption of 1 MW on its $1,920$ CPUs and 280 GPUs when playing the game with one of the best human players whose brain is rated at 20 W [Drubach, 2000]. Although we are still far from understanding the brain thoroughly, it is believed that the performance gap between computation in the biological nervous system and in a computer lies in the nature of the fundamental computing units and how they compute. Typical computers employ Boolean logic and deterministic digital operations usually based on synchronous clocks while nervous systems employ parallel, distributed, event-driven, stochastically unreliable components [Indiveri et al., 2009]: neurons. These impressive disparities in cognitive capabilities and energy consumption drive research into biologically-plausible spiking neurons and brain inspired computers, known as Neuromorphic Engineering (NE).

NE was proposed by Carver Mead in the late 1980s [Mead, 1989] to build analogue

circuits which mimic biological neural cells and the architecture of the nervous system using Very-Large-Scale Integration (VLSI) technology. With the ultimate goal of equipping neuromorphic machines with genuine intelligence, the objectives of NE can be summarised as follows [Furber and Temple, 2007]:

- brain modelling: for neuroscientists to understand the brain by modelling and simulating the activities of biological neurons;
- neuromorphic computing: for engineers to build brain-like machines by applying biological principles to computers.

The aims complement each other; building a biologically inspired computer requires a better understanding of the brain, and simulating brain activities at large scale and in real time is feasible only on massively-parallel neuromorphic hardware.

Spiking Neural Networks (SNNs), comprised of spiking neurons, hold the key to address the dual aims of understanding brain functions and building brain-like machines. The spiking neuron mathematically models the dynamics of a single neuron with biological realism and the network describes the architecture of the neural connections and the information transmission among them; readers can refer to Chapter 2 for more detail. Therefore, neuroscientists are able to reproduce the recorded neural dynamics and activities from *in-vivo/vitro* experiments to verify their models and measure the progress of brain understanding, while computer engineers can focus on the hardware implementations of the spiking neurons and the interconnections between them to build energy-efficient neuromorphic hardware.

Over the last decade, considerable development has taken place in NE where simulations of massive SNNs [Markram, 2006; Ananthanarayanan et al., 2009] have proved to be significantly useful in understanding the brain, and large-scale neuromorphic platforms have been launched to simulate SNNs in hardware. These neuromorphic computers develop into energy-efficient systems by implementing neurons, synapses and neuronal communications on analogue circuits ~~[Schemmel et al., 2010; Benjamin et al., 2014;~~ exploiting parallel low-power microprocessors on digital hardware [Furber et al., 2014; Merolla et al., 2014]. Thus, the neuromorphic hardware systems ~~composed with those~~ have successfully demonstrated decreased energy cost of SNN simulations on supercomputers [De Garis et al., 2010; Sharp et al., 2012].

However, the SNN simulations only reconstruct the network behaviours and neural dynamics of some subsystem of the brain, 'but without precisely functionally simulating that subsystem' [De Garis et al., 2010]. In other words, the SNNs are able to repeat the firing activities of groups of neurons, however, not capable of simulating or understanding the functions of these activities. Therefore, this type of SNN simulation can be used to guide neuroscience and the development of neuromorphic hardware systems, but is not directly useful for solving cognitive tasks. Recent SNN applications [Bill and Legenstein, 2014; Diehl et al., 2015a] in Artificial Intelligence (AI) tasks, summarised in Chapter 6, typically comprise only two neural layers and exploit biologically-plausible learning rules, e.g. Spike-Timing-Dependent Plasticity (STDP), and/or Winner-Take-All (WTA) circuits on the synaptic connections. These two-layered SNN models are considered to be 'reactive' since the output neurons simply react to the sensory input. Consequently, such SNNs cannot perform sophisticated effective cognition as can the brain; thus programming these neuromorphic machines to be competent in cognitive applications still remains unsolved. Indiveri et al. [2009] argued that the next substantial challenge of NE is to make these brain-like computers effectively cognitive, also known as 'Neuromorphic Cognition'.

STDP as a learning mechanism based on biological observations has been ~~theoretically proved~~ implemented to be equivalent to a stochastic version of powerful machine learning algorithms, such as Expectation Maximisation [Nessler et al., 2013], Contrastive Divergence [Neftci et al., 2013], Markov Chain Monte Carlo [Buesing et al., 2011] and Gradient Descent [O'Connor and Welling, 2016]. However, in practice, there have been two significant problems prohibiting the SNN from becoming as 'intelligent' as its non-spiking counterpart, the Artificial Neural Network (ANN). Firstly, Deep Learning research has made great achievements in the field of ANNs and dominated state-of-the-art solutions for AI engineering tasks, e.g. exceeding human-level performance on image classification [He et al., 2015], see Chapter 3 for more examples. However, the fundamental differences in data representation and neural computation between spiking and artificial neurons make it difficult to transform ANN models into SNN algorithms, see Chapter 2 for more detail. Secondly, the ~~computation~~ computational cost for simulating large SNNs of size comparable to commonly-used deep ANNs was considered to be infeasible, though this has gradually been solved by NE.

With the neuromorphic platforms ready for massive SNN simulations, this, therefore, is the main research problem: to improve the cognitive performance of SNNs to catch up with that of ANNs. Hence, researchers turn to Deep Learning to build 'smarter' SNNs. Initial studies have shown that SNNs can be trained by first training an equivalent deep ANN and then transferring the tuned weights to the SNN; this method is called 'off-line' training, since it does not take place on SNNs directly, but rather on ANNs instead. Chapter 4 discusses these 'off-line' training models in detail, and proposes a simple, generalised, off-line SNN training method to overcome the problems of poor modelling accuracy and high computational complexity of the existing methods [Jug et al., 2012; Hunsberger and Eliasmith, 2015; Diehl et al., 2015b]. To embed the biologically-plausible learning rules into deep SNN training, researchers take an extra step to 'on-line' methods where Deep Learning modules can be trained purely on SNNs in an event-driven manner, see Chapter 5. Previous work [Neil, 2013; Neftci et al., 2013; Burbank, 2015] has failed to provide SNNs with recognition accuracy equivalent to ANNs due to the lack of ~~mathematical analysis~~model formalisation and accurate parameter settings. We continue the inspiring work on these biologically-plausible 'on-line' training methods and propose a formalised method to train multi-layered Deep Learning modules on SNNs.

To provide meaningful comparisons between these proposed SNN models and other existing methods within the rapidly advancing field of NE, we propose a large dataset of spike-based ~~visual stimuli~~ images/videos to unify data resources for objective comparisons; and a corresponding evaluation methodology to estimate the overall performance of SNN models and their hardware implementations in Chapter 6. Moreover, we transform one of the common datasets widely used in Computer Vision into spike-based dataset to enable meaningful comparisons between SNNs and conventional machine learning methods.

1.1 Motivation and Aims

NE has led to the development of biologically-inspired computer architectures ~~which may~~ whose long-term goal is to approach the performance of the human brain in terms of energy efficiency and cognitive capabilities. Although there are a number

of neuromorphic platforms available for large-scale SNN simulations, the problem of programming these brain-like machines to be competent in cognitive applications still remains unsolved. On the other hand, Deep Learning has emerged in ANN research to dominate state-of-the-art solutions for cognitive tasks. Thus the main research problem emerges of understanding how to operate and train biologically-plausible SNNs to close the gap in cognitive capabilities between SNNs and ANNs on AI tasks.

Enabling this massively-parallel neuromorphic hardware to deliver state-of-the-art performance on AI tasks will be a big step towards Neuromorphic Cognition. It will contribute to the ultimate goal of equipping brain-inspired computers with Human brain levels of energy efficiency and cognitive capability.

1.2 Thesis Statement and Hypotheses

Although fundamental differences in input/output representation and neural computation exist between spiking and conventional artificial neurons, the cognitive capability of SNNs can be improved to catch up with that of ANNs by embedding Deep Learning techniques in training SNNs. Deep Learning has not only successfully equipped ANNs with better-than-human performance on AI tasks, but also ~~theoretical~~ studies have proved the equivalent learning capability of SNNs, and neuromorphic hardware is ready for operating large-scale deep SNNs.

According to the thesis statement, the hypotheses are defined as follows:

- Deep SNNs can be successfully and simply trained off-line where the training takes place on equivalent ANNs and the tuned weights then transferred back to the SNNs, thus making them as competent as ANNs in cognitive tasks.

- Unsupervised Deep Learning modules can be trained on-line on SNNs with biologically-plausible synaptic plasticity to demonstrate a learning capability equivalent to ANNs.

- A new set of spike-based vision datasets can provide resources and corresponding evaluation methodology to support objective comparisons and measure progress within the rapidly advancing field of NE.

1.3 Contributions

The primary achievement of the work described in this thesis is the training of deep SNNs, both off-line and on-line, which closes the gap in cognitive capability between SNNs and ANNs. Other achievements contribute to the performance evaluation of SNN models and their hardware implementations. The contributions are:

- **A generalised and simple method for off-line SNN training.**

  The core elements of the training methods are a pair of novel activation functions used in ANNs: Noisy Softplus (NSP) and the Parametric Activation Function (PAF). NSP successfully models the firing activities of biologically-plausible spiking neurons with conventional activation functions of abstract values; and PAF maps these numerical values to concrete physical units in SNNs: current in nA and firing rates in Hz. The proposed activation functions solve the problem of the fundamental differences in data representation and neural computations between ANNs and SNNs, thus tackle the difficulties of transforming ANN models to SNNs. Moreover, they address the problems of inaccurate modelling and high computational complexity of existing approaches.

  This off-line training method consists of three simple steps: firstly, estimate parameter p for the PAF, $y = p \times f(x)$, using the proposed activation function NSP; secondly, use a PAF version of conventional activation functions, e.g. Rectified Linear Unit (ReLU), for ANN training; thirdly, the tuned weights can be transferred directly into the SNN without any further transformation. This method involves the least computational complexity while performing most effectively among existing algorithms.

  NSP is described in Chapter 4 and was published and presented at the International Conference on Neural Information Processing (ICONIP 2016); the work of generalised SNN training using PAF ~~has been~~ will be submitted to the ~~Annual Conference on Neural Information Processing Systems (NIPS 2017~~IEEE Transactions on Neural Networks and Learning Systems (INNLS).

- **An on-line unsupervised learning algorithm working purely on event-based STDP for training spiking Autoencoders (AEs) and Restricted**

Boltzmann Machines (RBMs).

Multiplying two numerical values, which is the core operation in the algorithms for training the Deep Learning modules of AEs and RBMs, can be represented with rate multiplication of a pair of rate-coded spike trains. The proposed formalised Spike-based Rate Multiplication (SRM) method transforms the product of rates to the number of coincident spikes emitted from a pair of connected spiking neurons, and the simultaneous events can be captured by the change of the synaptic efficacy using the biologically-plausible learning rule: STDP.

The SRM successfully tackles the problem of translating the weight tuning from numerical computations to event-based, biologically-plausible learning rules in SNNs. In addition, the numerical analysis of the proposed algorithm accurately estimates the parameters, thus closely mimicking the learning behaviour of the AE and RBM modules, and improves the learning performance compared to existing methods. Moreover, we propose solutions to the problem of continuous performance drop caused by correlated spike trains. Thus, spiking AEs and RBMs can be trained with SRM and approach the same, sometimes even superior, classification and reconstruction capabilities compared to their equivalent non-spiking models.

This work comprises Chapter 5. A paper on these findings is in preparation for submission to the Journal of Neural Computation.

- **A dataset and the corresponding evaluation methodology for comparisons of SNN models and their hardware implementations.**

  To objectively compare these proposed SNN models with other existing methods, we propose a Neuromorphic Vision dataset NE15-MNIST which is comprised of ~~spike-based visual stimuli~~ spike-encoded images/videos based on a standard computer vision benchmark, the MNIST [LeCun et al., 1998] dataset. The unified dataset satisfies the requirement for quantitatively measuring progress within the rapidly advancing field of NE and provides resources to support objective comparisons between researchers. In addition, a complementary evaluation methodology is presented to estimate the overall performance of SNN models and their hardware implementations, since new concerns relating to energy efficiency and

recognition latency emerge in SNNs run on NE platforms.

We also present a potential benchmark system which is evaluated using the Poissonian subset of the NE15-MNIST dataset. It provides a baseline for further comparisons with upcoming SNN models.

The dataset was generated with the help of Garibaldi Pineda-García and Teresa Serrano-Gotarredona. This work comprises Chapter 6 and was published as a journal paper in Frontiers in Neuromorphic Engineering.

1.4 Papers and Workshops

1.4.1 Papers

Much of the work contributed to solving the main research problem of this thesis has either been published or is in the process of submission for publication.

- **Q. Liu**, and S. Furber, **Noisy Softplus: A Biology Inspired Activation Function**, International Conference on Neural Information Processing (ICONIP 2016). This paper [Liu and Furber, 2016] introduces the novel activation function, NSP, which solves the problem of accurately modelling the response firing activity of spiking neurons using conventional abstract activation functions. This paper comprises the first half of Chapter 4.

- **Q. Liu**, Y. Chen, G. García, and S. Furber, **Generalised Training of Spiking Neural Networks**, (~~submitted to NIPS 2017~~to be submitted to INNLS). This paper extends the work of the NSP to solve the problem of mapping abstract numerical values of activation functions to concrete physical units in spiking neurons using PAF, and successfully formalises a simple off-line SNN training method which is also generalised to ReLU-like activation functions. The paper presents the work described in the rest of Chapter 4.

- **Q. Liu**, and S. Furber, **Spike-based Rate Multiplication for On-line SNN Training** (to be submitted to Neural Computation). This paper mainly comprises the work of Chapter 5, which proposes a method for on-line unsupervised

training of SNNs equivalent to the conventional Deep Learning techniques: AEs and RBMs.

- **Q. Liu**, G. García, E. Stromatias, T. Gotarredona, and S. Furber, **Benchmarking Spike-Based Visual Recognition: A Dataset and Evaluation**, Frontiers in Neuromorphic Engineering. The work presented in this paper [Liu et al., 2016] mainly comprises the spike-based dataset **NE15-MNIST** and its corresponding evaluation method for Neuromorphic Vision proposed in Chapter 6. In addition, the paper also includes the contributions of the co-authors: the detailed description of a subset of this database and a case study as an example to validate the dataset and its evaluation.

Other publications build up the neuromorphic hardware system for complete event-based visual and auditory processing, providing deep spiking neural networks a valid hardware platform for running applications in biological real time.

- **Q. Liu**, and S. Furber, **Real-Time Recognition of Dynamic Hand Postures on a Neuromorphic System**, International Conference on Artificial Neural Networks (ICANN 2015). We develop an object recognition system operating in real-time on a complete neuromorphic platform in an absolute spike-based fashion. This paper paves the way for further study with solid proof of the capability of a real-time cognitive application built on a neuromorphic platform. In Chapter 2, we introduce this system as an existing vision-based neuromorphic hardware platform which comprises a Dynamic Vision Sensor (DVS) as the front-end and a massive-parallel SNN hardware simulator as the back-end.

- **Q. Liu**, C. Patterson, S. Furber, Z. Huang, Y. Hou and H. Zhang, **Modeling Populations of Spiking Neurons for Fine Timing Sound Localization**, International Joint Conference on Neural Networks (IJCNN 2013). This paper [Liu et al., 2013] presents a model of sound localisation to solve the problem of coarse time resolution of SNN simulations. Such an auditory processing system can be implemented on a similar neuromorphic hardware platform described above, which uses a silicon cochlea as the input (see Chapter 2).

- G. García, P. Camilleri, **Q. Liu**, and S. Furber, **pyDVS: An Extensible, Real-time Dynamic Vision Sensor Emulator using Off-the-Shelf Hardware**,

The 2016 IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2016). This paper [Garibaldi et al., 2016] proposes a visual input system inspired by the behaviour of a DVS but using a conventional digital camera as a sensor and a PC to encode the images (see Chapter 2).

1.4.2 Workshops

The author participated in workshops organised by the NE community, 1) to establish and contribute to collaborations on mutual interests; 2) to catch up with cutting-edge research and collect inspiration; and 3) to discuss the author's own findings with key researchers in the field.

- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2012.*

  Contributed to successful connections of SpiNNaker to neuromorphic sensors[1]. This formed the hardware platform for real-time SNN applications processing event-based sensor data.

- *Telluride Neuromorphic Cognition Engineering Workshop 2013.*

  Developed a real-time sound localisation system on the neuromorphic platform as a main contributor[2]. The work led to the publication of a journal paper [Lagorce et al., 2015].

- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2014.*

  Developed the real-time neural activity visualiser for the project of 'Integrated Neurorobotics for Real-World Cognitive Behaviour' [3].

- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2015.*

  Inspired by the projects on Deep Learning in the workshop[4], the author later proposed the off-line SNN training method and the unsupervised on-line learning algorithm of deep SNNs, and led the discussion of benchmarking neuromorphic vision in the workshop [5].

[1] `https://capocaccia.ethz.ch/capo/wiki/2012/csnQian`
[2] `http://neuromorphs.net/nm/wiki/sound_localization`
[3] `https://capocaccia.ethz.ch/capo//wiki/2014/integrneurobot14`
[4] `https://capocaccia.ethz.ch/capo//wiki/2015/spikednn15`
[5] `https://capocaccia.ethz.ch/capo//wiki/2015/visionbenchmark15`

1.5 Thesis Structure

The thesis comprises the following seven chapters:

Chapter 1 introduces the origin and the motivation of the research, states the problem, defines the hypotheses and objectives, summarises the contributions and publications, and outlines the thesis.

Chapter 2 illustrates how biological neurons function, transmit signals between them, and are modelled by mathematical abstractions, thus to unveil the special features of spiking neurons that differ from the neurons of ANNs; and introduces SNN simulators both in software and in hardware including neuromorphic systems.

Chapter 3 gives an overview of popular architectures and models of Deep Learning and illustrates the mechanism of the Convolutional Networks (ConvNets), the AEs, and the RBMs in detail.

Chapter 4 demonstrates the generalised off-line SNN training method to confirm the first hypothesis that SNNs can be trained off-line and perform equivalently as ANNs in cognitive tasks.

Chapter 5 proposes an STDP-based learning algorithm for training spiking AEs and RBMs on-line; and test the second hypothesis that on-line training is able to improve the cognitive capabilities of SNNs and catch up with ANNs.

Chapter 6 puts forward the spike-based vision dataset and the evaluation methodology and presents a case study as a tentative benchmark running on SpiNNaker to assess the hardware-level performance against software simulators.

Chapter 7 summarises the research, discusses the contributions to the field, points out future directions and concludes the thesis.

1.6 Summary

In brief, Figure 1.1 summarises the introduction and demonstrates the outline of the thesis.

Chapter 1 introduces the aims of NE: modelling the brain, and building brain-like machines. Although progress has been made in both directions, it is still far from achieving the long term goal of Neuromorphic Cognition. With the support of accumulated knowledge of SNNs and the massive neuromorphic SNN simulators (both

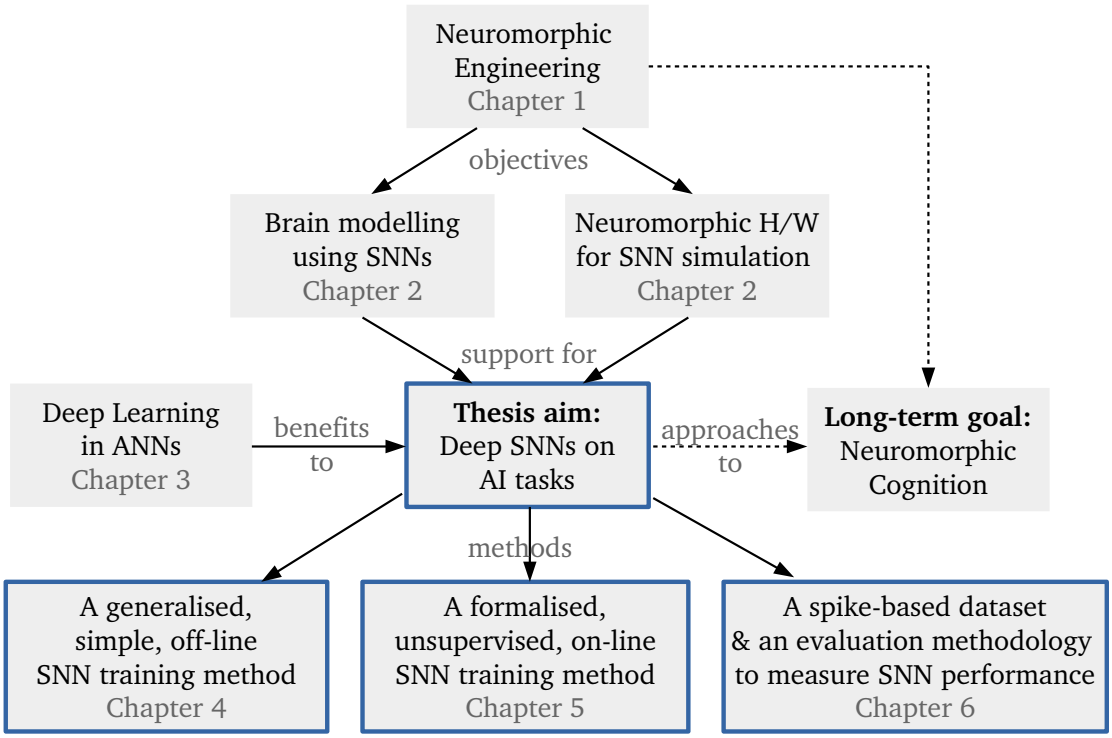


Figure 1.1: The outline of the thesis.

refer to Chapter 2), plus the huge success of Deep Learning in ANNs (see Chapter 3), the research aims at understanding how to operate and train biologically-plausible SNNs to close the gap in cognitive capabilities between SNNs and ANNs on AI tasks, thereby approaching Neuromorphic Cognition.

To achieve the thesis aim, we propose an off-line (Chapter 4) and an on-line (Chapter 5) SNN training method to bring Deep Learning advantages to SNNs, and provide a spike-based dataset and its corresponding evaluation methodology to measure the performance of SNN models and the neuromorphic hardware platforms in Chapter 6.

Chapter 2

Spiking Neural Networks (SNNs)

The so-called third generation of artificial neural network, the Spiking Neural Network (SNN), is comprised of spiking neurons which mimic the dynamics of biological neural behaviour. In this chapter we will demonstrate the special features of spiking neurons that differ from neurons of conventional Artificial Neural Networks (ANNs); these biologically-plausible neuronal operations are the root of the research problem raised in the thesis: how to operate SNNs to equip them with cognitive capabilities as competent as ANNs. Section 2.1 will illustrate how biological neurons function and transmit signals between them. The way neural dynamics are modelled by mathematical abstractions of spiking neurons is described in Section 2.2 , and finally existing SNN simulators are introduced in Section 2.3.

2.1 Biological Neural Components

At the cellular level, the central nervous system consists of two types of cell: neurons, the elementary processing units, and glial cells, the structural and metabolic supporters. Here we focus on the former, since neurons are the basic elements supporting higher brain functions such as cognition, thought and action. The human brain contains around a hundred billion (10^{11}) such processing units, and up to four orders of magnitude more connections (10^{15})~~, all over the brain~~ [Azevedo et al., 2009]. Despite being such a ~~comprehensive~~ huge and complex system, neurons in the brain manage to send signals rapidly and precisely to other cells through these connections, thanks to their special structures.

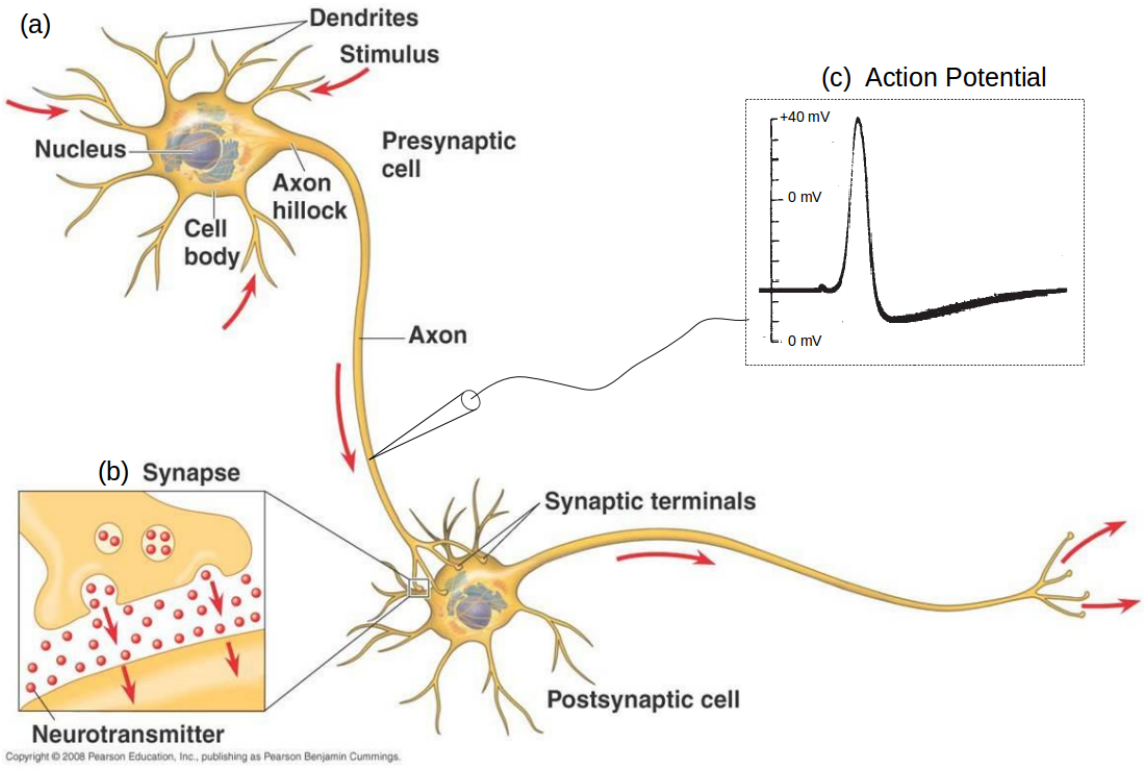


Figure 2.1: Two neurons connected by synapses. A neuron comprises three functional parts: dendrites, the cell body, and the axon. (a) A pre-synaptic cell connects to its post-synaptic cell through synapses (b) [Reece et al., 2011], and the neural signal, the action potential (c) [Hodgkin and Huxley, 1939], propagates in the direction of the red arrows.

2.1.1 Neuron

A typical neuron comprises three functional parts: dendrites, a cell body (soma), and an axon, see Figure 2.1(a). The dendrites of a neuron receive stimuli from other neurons, and transmit the neural signal to the neuron's soma. The soma is the cell body of the neuron, the location of the nucleus, and functions as a non-linear processor which triggers an output signal when the accumulated total input exceeds some threshold. The output signal initiates from the axon hillock where the axon emerges from the soma, and is propagated through the axon to other neurons. Most neurons have only one axon, but may connect to many neurons by branching out axon terminals.

The signal delivery from one neuron to another occurs at the junction between these two neurons, which is called a synapse, see Figure 2.1(b). The configuration can be seen as a pre-synaptic cell which sends the signal, and a post-synaptic cell which receives it.

2.1.2 Neuronal Signals

Neuronal signals propagated among neurons are short electrical pulses, and Figure 2.1(c) shows the original recording of such a so-called **action potential** observed on a squid giant axon. A typical action potential, also known as a '**spike**', is of about 100 mV amplitude and lasts 1-2 ms. Usually, there is a time period immediately after a spike that the neuron is unresponsive to any further stimulus. This minimal time difference between two spikes of a single neuron is the absolute **refractory period** during which no spike can be generated. After the absolute refractory period, it is still difficult but possible to fire a spike during the relative refractory period.

The size and duration of the spikes do not vary much among different species, and maintain the same form as the electrical pulses propagate along the axon as illustrated in Figure 2.1(c). Therefore, the form of an action potential carries little information; it is the frequency and timing of the spikes that encode the messages. A sequence of action potentials generated by a single neuron is called a '**spike train**', which can be viewed as binary events happening in discrete time where 'on' indicates a spiking event within a time step whereas 'off' means none. Information can be encoded in the frequency and timing of these binary events.

The rate coding model states that the spiking rate represents the intensity of a stimulus, e.g. as the stimulus becomes stronger, the frequency of the action potentials also increases. An example of the tuning curve of a V1 (visual area one of the visual cortex) simple cell responding to different stimulus orientation is shown in Figure 2.2. As the stimulus becomes more aligned to the preferred orientation (0°) of the neuron, the firing rate increases.

Rate coding works well when the stimulus is changing slowly and the observation time period is long enough to estimate the firing rate. However, in practice the stimulus, e.g. visual sensory input, varies on a fast time scale and the neurons respond within a short reaction time. Thus, temporal coding encodes information in the precise timing of spikes which is considered to be a candidate for the encoding of a fast changing stimulus.

Sound localisation requires temporal coding at sub-millisecond precision, which is a good example of one of the temporal coding schemes, phase locking. Figure 2.3 shows phase-locked spike trains generated by Inner Hair Cells in the cochlea. Phase

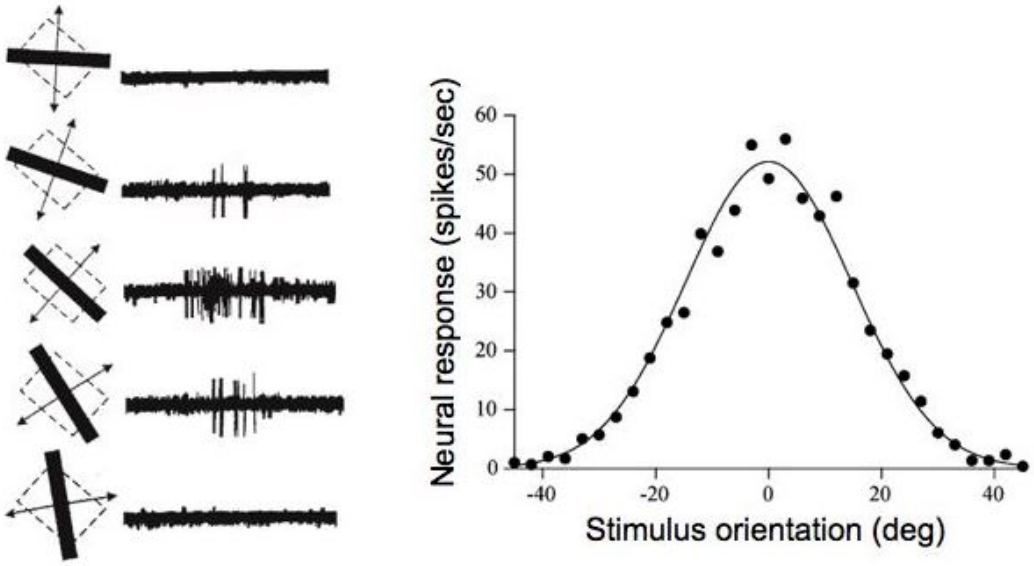


Figure 2.2: Example of rate coding: spike trains for different stimulus orientation (left) of a V1 simple cell of a cat, and the tuning curve (firing rate against stimulus orientation) of the neuron (right) [Hubel and Wiesel, 1962]. The square indicates the visual receptive field of the neuron, and a bar is placed at different orientations and moves to the direction perpendicular to its orientation. As the stimulus becomes more aligned to the preferred orientation (0°) of the neuron, the firing rate increases.

locking forms the basis of detecting time differences of binaural sound inputs.

Time-to-first-spike encodes the information according to the intensity of a stimulus where a spike shortly after a reference signal indicates a strong stimulation and a later action potential is interpreted as a weaker input. The tactile afferent information generated by forcing fingertips from various directions is encoded in such a time-to-first-spike coding scheme [Johansson and Birznieks, 2004]. Synchrony coding also can be found in the brain, where neurons representing the same 'concept' always fire at the same time [Von Der Malsburg, 1994], for example in object recognition [Gray and Singer, 1989]. Established from the context of fast object recognition, rank-order coding was proposed where the precise time of spikes is discarded, but rather uses the relative order of spikes among a group of neurons [Gautrais and Thorpe, 1998].

2.1.3 Signal Transmission

The spike, as an electrical signal, propagates to another neuron through the junction between these two neurons, a chemical synapse. The axon terminal of a pre-synaptic neuron approaches very close (within about 20 nm) to the dendrites (or cell body) of

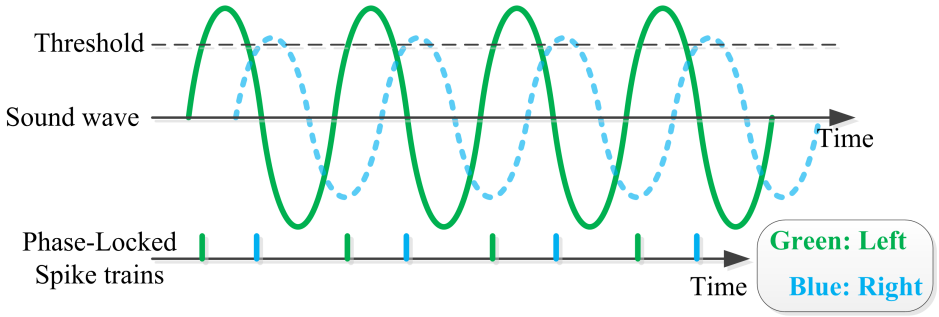


Figure 2.3: Example of temporal coding: phase-locked spike trains generated by simulated Inner Hair Cells in the cochlea [Liu et al., 2013]. A sound source generates a sine wave of a certain frequency and conducts to two ears with a time difference and different amplitude due to the angle and distance of the sound source to the head. Two spike trains respond to different phases manipulated by a threshold of the sound waves. Sound localisation can be resolved by calculating the time difference and/or level difference of these sound waves which are encoded in the spike trains.

a post-synaptic neuron. The tiny space between neurons at a synapse is called the synaptic cleft, which is illustrated in Figure 2.1(b). At such a chemical synapse, the action potential generated by the pre-synaptic neuron triggers chemical neurotransmitter molecules to be released into the synaptic cleft, and once the post-synaptic neuron detects these neurotransmitters it opens specific ion channels to allow electrical current in. Hence, synapses complete the transformation from electrical signal to chemical molecules and then back to ion influx. The amount of neurotransmitter determines the strength of the current flow into the post-synaptic neuron. Thanks to synaptic plasticity, changes of chemical synapses enable modulations of the synaptic efficacy, and form the neuronal correlation of learning and memory.

2.2 Modelling Spiking Neurons

2.2.1 Neural Dynamics

The effect of an ion influx on the post-synaptic neuron caused by spike transmission is a change of potential difference between the interior and exterior of the cell body, which is called the **membrane potential**. The membrane potential of a post-synaptic neuron stays at a **resting potential** in the absence of an input. As soon as a spike arrives, the membrane potential will be either depolarised (increased) or hyper-polarised (decreased) according to the type of synapse, and go back to the resting potential

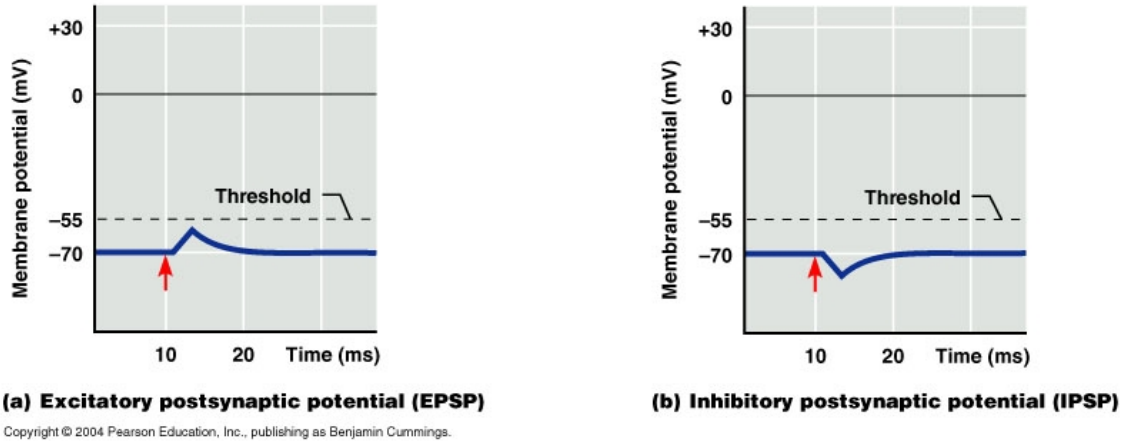


Figure 2.4: Post-synaptic potential driven by a spike, where the red arrow represents a spike arriving at the neuron [Marieb and Hoehn, 2007].

driven by membrane leakage. The state of the membrane potential change caused by a single spike is called the Post-Synaptic Potential (**PSP**). Thus, a spike transmitted by an excitatory synapse triggers a positive PSP, called an Excitatory Post-Synaptic Potential (**EPSP**), see Figure 2.4(a); a negative change, an Inhibitory Post-Synaptic Potential (**IPSP**), is driven by an inhibitory synaptic event and is shown in Figure 2.4(b). Spikes arriving at different synapses at the same post-synaptic neuron have PSPs of different amplitudes according to the synaptic efficacy.

Multiple PSPs have an accumulative effect on the membrane potentials both in

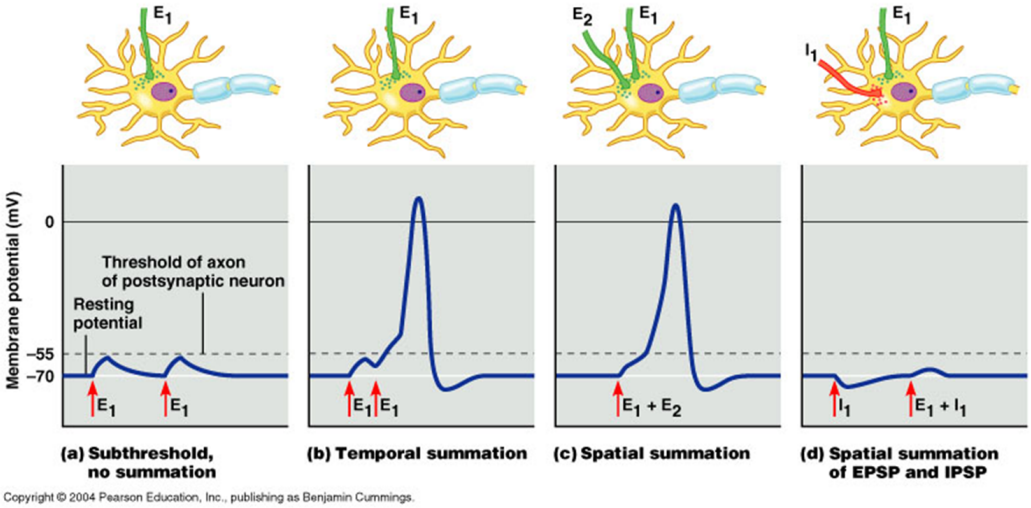


Figure 2.5: Summation of post-synaptic potentials [Reece et al., 2011]. (a) Single EPSPs are usually not strong enough to trigger an action potential without summation. (b) Temporal summation of two EPSPs of the same synapse generates an action potential. (c) Spatial summation of two EPSPs of two synapses generates an action potential. (d) Spatio-temporal summation of both EPSP and IPSPs.

temporal and spatial terms. The accumulation performs a simple summation of PSPs until the membrane potential reaches a **threshold**, when an action potential is generated at the post-synaptic neuron. Figure 2.5 illustrates temporal and spatial summations of PSPs under different circumstances. The temporal summation refers to the accumulated effect of a single synapse where the spatial one integrates the PSPs triggered by multiple synapses.

The neural dynamics of the membrane potentials, PSPs, and spike trains are all time dependent, while the neurons of ANNs, e.g. sigmoid units, only cope with numerical values representing spiking rate, without timing information, see Figure 2.6. A regular artificial neuron (Figure 2.6(a)) comprises a weighted summation of input data, $\sum x_i w_i$, and an activation function, f, applied to the sum. Usually, a bias is included in the weighted summation which ~~can be seen as an extra input $x_b = 1$ with its weight set to b~~increases the expression ability of a neuron. However, in this thesis we ~~exclude biases for both artificial and spiking neurons~~ remove biases of both ANNs and SNNs to simplify the neural models and to reduce the number of parameters. Nevertheless, our experimental results show that the performance almost keeps the same when solving a relatively simple task, the MNIST. Thus the inputs of a spiking neuron (Figure 2.6(b)) are spike trains generated by pre-synaptic neurons, which create PSPs on the post-synaptic neuron and trigger a spike train as the output of this spiking neuron. These fundamental differences in input/output representation and neural computation lead to special model descriptions of spiking neurons (illustrated in the next section), and raise the difficulties of transforming ANN models to spiking neurons. Hence, this research aims to address the problem of how to operate and train biologically-plausible SNNs to be as competent in cognitive tasks as are ANNs.

2.2.2 Neuron Models

The keys to modelling a spiking neuron are:

- to mathematically formalise the evolution of the membrane potential;
- to state a mechanism of spike generation.

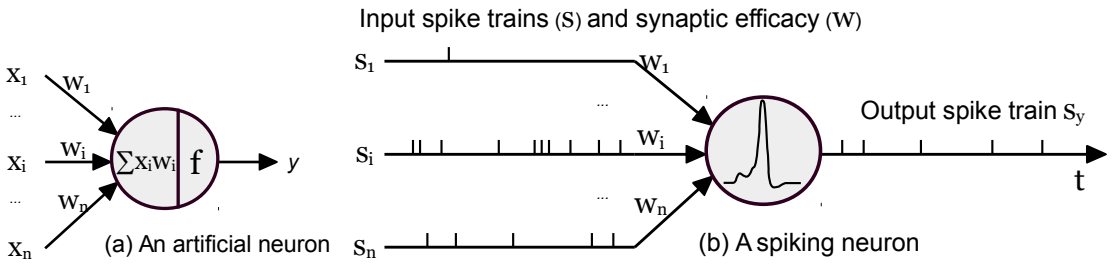


Figure 2.6: Comparisons of processing mechanisms of an artificial and a spiking neuron. (a) An artificial neuron takes numerical values of vector $\mathbf{x}$ as input, works as a weighted summation followed by an activation function f. (b) Spike trains flow into a spiking neuron as input stimuli, trigger linearly summed PSPs through synapses with different synaptic efficacy $\mathbf{w}$, and the post-synaptic neuron generates output spikes when the membrane potential reaches some threshold.

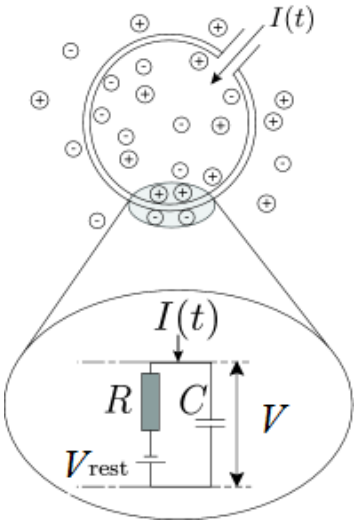


Figure 2.7: The cell membrane acts like a Resistor-Capacitor (RC) circuit [Gerstner et al., 2014]. Input current $I(t)$ flows into a neuron which charges the capacitor C and leaks through the resistance R in line with a battery V_{rest}.

Leaky Integrate-and-Fire (LIF) Model

The evolution of membrane potential V can be simplified to a Resistor-Capacitor (RC) circuit which consists of a membrane ~~capacitor~~ capacitance C_m and a membrane resistance R_m, both driven by an input current flow I, see Figure 2.7. In the resting state without any input, the membrane potential V stays at the same potential as the battery V_{rest}. When current flows into the neuron, it will charge the capacitor with current $I_C(t)$ and discharge through the resistance with current $I_R(t)$. When the input current stops the capacitive charge will decay back to V_{rest} by leaking through

the resistance:

$$\begin{aligned} I(t) &= I_R(t) + I_C(t) \\ &= \frac{V - V_{rest}}{R_m} + C_m \frac{\mathrm{d}V}{\mathrm{d}t} \quad . \end{aligned} \tag{2.1}$$

The standard form of the LIF model describes the sub-threshold membrane potential evolution as follows:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - V_{rest}) + R_m I(t) \ , \tag{2.2}$$

where $\tau_m = C_m R_m$ is called the membrane time constant, and as soon as the membrane potential reaches the threshold V_{thresh}, it is set to a reset potential V_{reset}, which is usually lower than V_{rest}:

$$V = V_{reset} \ . \tag{2.3}$$

The simple LIF model uses: (1) a linear differential equation to describe the evolution of membrane potential; and (2) a threshold to generate a spike.

Hodgkin-Huxley Model

The Hodgkin-Huxley model [Hodgkin and Huxley, 1952] is the Nobel Prize winning model that explains the ionic mechanisms generating and transmitting action potentials in the squid giant axon. The current $I_R(t)$ which flows through the membrane resistance is determined by three ion channels: a leak channel with conductance g_L, the sodium channel with conductance g_{Na} and the potassium channel with conductance g_K. The currents which flow through these channels are all proportional to the difference between the membrane potential and the reversal potentials of the channels: $V - E_L$, $V - E_{Na}$, and $V - E_K$ respectively. Thus Equation 2.1 is detailed as:

$$\begin{aligned} I(t) &= I_L(t) + I_{Na}(t) + I_K(t) + I_C(t) \\ &= g_L(V - E_L) + g_{Na} m^3 h (V - E_{Na}) + g_K n^4 (V - E_K) + C_m \frac{\mathrm{d}V}{\mathrm{d}t} \quad . \end{aligned} \tag{2.4}$$

The Hodgkin-Huxley model can be seen as a non-linear differential equation with four state variables, V, m, h and n that change against time:

$$\begin{aligned} C_m \frac{\mathrm{d}V}{\mathrm{d}t} &= I(t) - g_K n^4 (V - E_K) - g_{Na} m^3 h (V - E_{Na}) - g_L (V - E_L) \ , \quad \text{and} \\ \frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1 - m) - \beta_m(V) m \ , \\ \frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1 - n) - \beta_n(V) n \ , \\ \frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1 - h) - \beta_h(V) h \ , \end{aligned} \tag{2.5}$$

where $\alpha(V)$ and $\beta(V)$ are empirical functions of membrane potential.

With regard to the mechanism of spike initiation, the most significant property of the Hodgkin-Huxley model is that the model is able to generate action potentials with the changes of those dynamic internal variables alone.

The Hodgkin-Huxley equations provide a detailed, quantitative, and reasonably accurate mathematical model explaining the evolution of the membrane potential and the action potential [Byrne et al., 2014]. However, its numerical complexity and highly non-linear characteristics prohibit it from being intuitively understood and make large-scale simulations too expensive. Therefore, neural model selection should take account of objectives, degree of detail and computational power.

Izhikevich Model

The Izhikevich model was proposed to solve the problems of computational complexity of the Hodgkin-Huxley model and the insufficient capability of LIF model to reproduce the complex dynamics of cortical neurons [Izhikevich, 2003]. Thus the model can be employed to simulate large-scale brain models comprising real biological neurons.

The membrane potential evolves in accordance with a pair of differential equations:

$$\begin{aligned} \frac{\mathrm{d}V}{\mathrm{d}t} &= 0.04V^2 + 5V + 140 - u - I(t) \ , \\ \frac{\mathrm{d}u}{\mathrm{d}t} &= a(bV - u) \ , \end{aligned} \tag{2.6}$$

where ~~v~~ V is the membrane potential and u represents the membrane recovery which negatively feeds back to ~~v~~V.

In terms of spike generation, the initiation part of an activation potential is produced by the equations, but a resetting scheme is needed:

$$\begin{cases} V = c \\ u = u + d \end{cases} \quad \text{when } v \geq 30. \tag{2.7}$$

Parameters a, b, c, and d are constant, which can be configured to reproduce various neural dynamics of real biological neurons [Izhikevich, 2004].

2.2.3 Synapse Model

Applying spiking neuron models to synaptic spike transmission, we can use two types of synapse: current-based and conductance-based models. Thus the synaptic efficacy w determines either the input current intensity flowing through the synapse:

$$I(t) = w(t) \ , \tag{2.8}$$

or the electrical conductance g_{syn} of the ion channel:

$$I(t) = g_{syn}(V - E_{syn}) = w(t)(V - E_{syn}) \ , \tag{2.9}$$

where E_{syn} indicates the reversal potential of a synapse. Both equations identify the strength of a synaptic current, thus simply adding up all synaptic currents on the same post-synaptic neuron represents the external current $I(t)$ for all the neuron models stated in Section 2.2.2.

The current flow usually has a much longer time constant than an action potential and decays over time, thus a simple exponential decay is able to model the decaying synaptic efficacy. Assuming spikes are delivered at time $t = t_0, t_1, ..., t_n$, the initial synaptic weight is set to w_0 and τ_{syn} is the synaptic time constant, the decaying synaptic current or the conductance can be described as:

$$w(t) = \sum_k w_0 e^{-(t-t_k)/\tau_{syn}} \ . \tag{2.10}$$

In this thesis we mostly employ LIF neurons and a current-based synapse model with decaying synaptic efficacy, due to its simple mathematical expression, low numerical complexity and high-level abstraction hiding much of the detailed neural dynamics. Therefore, at the initial stage of merging artificial Deep Learning with biologically-plausible SNNs, we can (1) target standard LIF neurons which are supported by most

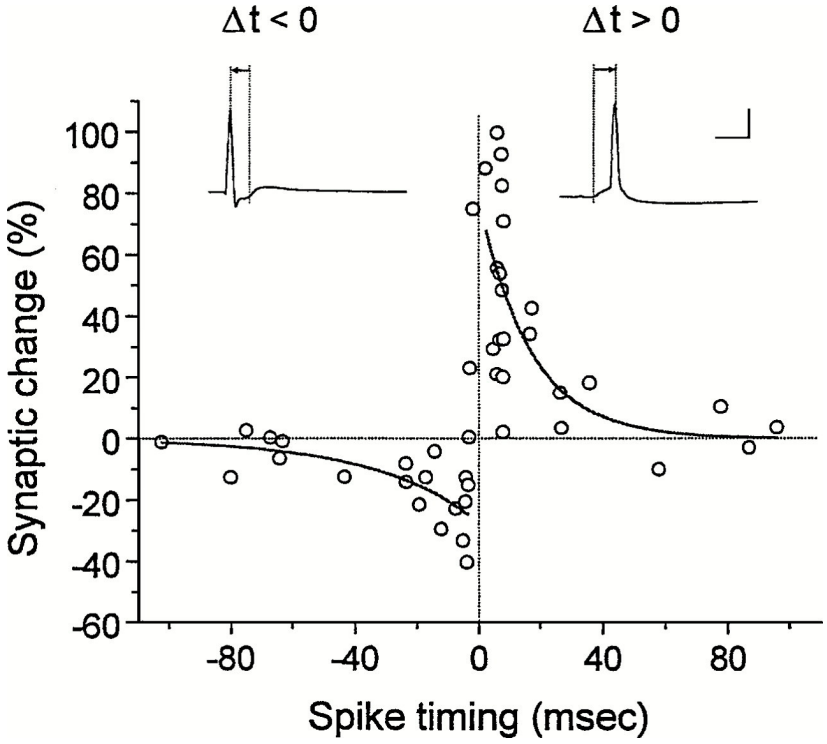


Figure 2.8: Spike-Timing-Dependent Plasticity (STDP) [Bi and Poo, 2001]. The circles record the synaptic weight change of real biological observations on 60 pairs of hippocampal neurons. Curves of exponential decays against relative timing of pre- and post-synaptic spikes fit well to the real biological data.

of the neuromorphic hardware systems; (2) simulate large-scale SNNs with deep architectures without a tight limitation on computational power; and (3) have fewer parameters thus resulting in a simplified problem.

2.2.4 Synaptic Plasticity

As mentioned in Section 2.1.3, synaptic plasticity provides the neuronal level of learning and the memory of the brain. Biological observations have provided evidence that modulations of the synaptic efficacy depend on the relative timing of the pre- and post-synaptic spikes [Bi and Poo, 1998]. This mechanism is known as Spike-Timing-Dependent Plasticity (**STDP**) [Song et al., 2000], and the standard STDP learning window is illustrated in Figure 2.8:

$$\Delta w = \begin{cases} A_+ e^{-\Delta t/\tau_+}, & \text{when } \Delta t \geq 0 \ , \\ -A_- e^{\Delta t/\tau_-}, & \text{when } \Delta t < 0 \ . \end{cases} \tag{2.11}$$

The synaptic weight is potentiated when a post-synaptic spike fires later than a pre-synaptic spike, and the amplitude of such a potentiation is determined by the curve of

exponential decay with a time constant τ_+ and an initial quantity A_+; however, when a post-synaptic spike is generated before a pre-synaptic one, synaptic depression will occur according to the exponential decay defined by τ_- and $-A_-$.

Besides the standard STDP model [Song et al., 2000], also known as the additive model, stated in Equation 2.11, variations of the STDP learning rule have been proposed to satisfy different learning speeds and classification accuracy. Multiplicative STDP [Morrison et al., 2008] is an alternative model where the weight change is not only dependent on the relative timing, but also on the current synaptic efficacy:

$$\begin{cases} A_+(w) = (w_{max} - w)\eta_+ \quad , \\ A_-(w) = w\eta_- \quad , \end{cases} \tag{2.12}$$

where A_+ and A_- in Equation 2.11 become functions of variables w, η_+ and η_- define the learning rate of the weight potentiation and depression, and w_{max} is the maximum synaptic efficacy while 0 is the minimum. We use the multiplicative STDP for SNN training in Section 6.5. It performs better than the ~~addictive~~ additive STDP when used in such a classification task, since the weight are much more distributed in the working range.

In Chapter 5 we exploit a much simplified version of asymmetric rectangular STDP where the weight change is just a constant within an STDP window, τ_{win}:

$$\Delta w = \begin{cases} \eta \ , & \text{when} \quad 0 \leq \Delta t \leq \tau_{win} \\ 0 \ , & \text{otherwise} \end{cases} \tag{2.13}$$

We choose this simple algorithm for the straightforward linear conversion of the number of coincident spikes to the weight update.

2.3 Simulating Networks of Spiking Neurons

In the previous section, we described neural dynamics as mathematical models at the neuronal level. However, it is challenging to simulate a large SNN with a high volume of synaptic connections, even using simple models such as LIF, because of the high event rate (10^4 synaptic events per second per single neuron on average). Addressing this problem, existing solutions vary from software simulators to neuromorphic hardware.

2.3.1 Software Simulators

Existing approaches to software simulation can be seen as: 'clock-driven' where the neural state is updated with some fixed time resolution, or 'event-driven' where the membrane potential is only modified when a spike arrives. The synchronous 'clock-driven' method uses numerical integration for solving the Ordinary Differential Equations (ODEs), that describe the evolution functions of the membrane potential with respect to time. However, with updates only on the time clocks (usually at 1 ms resolution), the non-linear differential equations can only be approximated rather than solved, and the spike times lose precision since they are bound to discrete time steps. 'Event-driven' approaches, in comparison, are accurate since they use explicit solutions of the ODEs and the spike arrival time is not rounded to time bins. Unfortunately, apart from LIF neurons, all the other models described in Section 2.2.2 are analytically unsolvable. The high synaptic event rate (10^4 Hz per neuron) takes no advantage of computational efficiency using this asynchronous approach. Therefore, most of the popular software simulators use a 'hybrid' solution, including NEST [Gewaltig and Diesmann, 2007] and Brian [Goodman and Brette, 2008], where neural state is updated synchronously, but the synapse operates in an event-based way.

Another software tool, PyNN [Davison et al., 2008], is a description language for building SNNs; it abstracts away the detail of various simulators and provides unified APIs for any simulator that supports it. Consequently, neuroscientists and SNN designers do not need to learn different 'languages' for specific simulators, and the models written in PyNN are supposed to run freely on the supporting simulators.

Most of the SNN models developed in this thesis are described in PyNN and run on NEST, and some of them are also tested on a hardware simulator, SpiNNaker [Furber et al., 2014], which will be introduced in the following section. In Chapter 5 we develop our own SNN simulator to implement and test a proposed learning algorithm, and the simulator follows the synchronous convention due to its programming simplicity and flexible neural model selection.

2.3.2 Neuromorphic Hardware

Neuromorphic systems can be categorised as analogue, digital, or mixed-mode analogue/digital, depending on how neurons, synapses and spike transmission are implemented. Some analogue implementations exploit sub-threshold transistor dynamics to emulate neurons and synapses directly in hardware [Indiveri et al., 2011] and are more energy-efficient while requiring less area than their digital counterparts [Joubert et al., 2012]. However, the behaviour of analogue circuits is hard to control through the fabrication process due to transistor mismatch [Indiveri et al., 2011; Pedram and Nazarian, 2006; Linares-Barranco et al., 2003], and achievable wiring densities render direct point-to-point connections impractical for large-scale systems.

The majority of mixed-mode analogue/digital neuromorphic platforms, such as the High Input Count Analog Neural Network (HI-CANN) [Schemmel et al., 2010], Neurogrid [Benjamin et al., 2014], and HiAER-IFAT [Yu et al., 2012], use analogue circuits to emulate neurons and digital packet-based technology to communicate spikes using Address-Event Representation (AER) [Lazzaro and Wawrzynek, 1995] protocol. This enables reconfigurable connectivity patterns between the neurons and fulfils the real-time requirement.

Digital neuromorphic platforms such as TrueNorth [Merolla et al., 2014] use digital circuits with finite precision to simulate neurons in an event-driven manner to minimise the active power dissipation. Such systems suffer from limited model flexibility, since neurons and synapses are fabricated directly in hardware with only a small subset of parameters under the control of the researcher. The SpiNNaker many-core neuromorphic architecture [Furber et al., 2014] uses low-power programmable cores and scalable event-driven communications hardware allowing neural and synaptic models to be implemented in software. While software modelling provides great flexibility, digital platforms generally have reduced precision (due to the inherent discretisation) and higher energy consumption when compared to analogue platforms.

2.3.3 Neuromorphic Sensory and Processing Systems

Neuromorphic engineers have successfully produced visual and auditory silicon devices mimicking the biological retina and cochlea, and boosted the applications of spike-based sensory processing in artificial vision and audition.

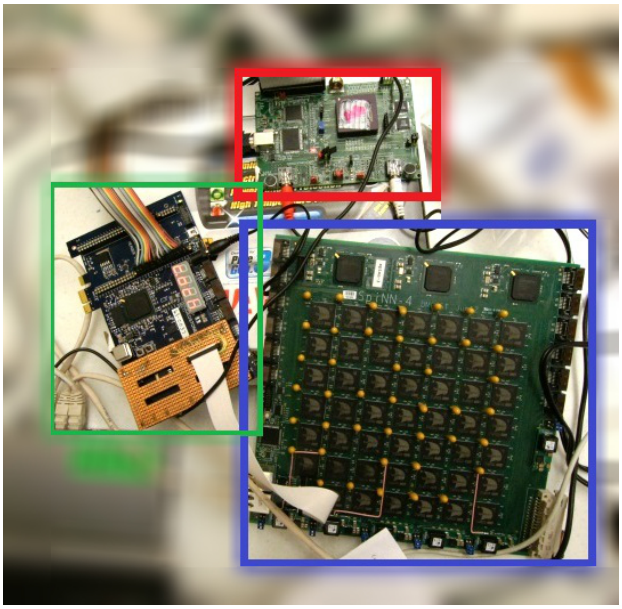
The visual input is captured by a DVS (Dynamic Visual Sensor) silicon retina [Delbruck, 2008; Serrano-Gotarredona and Linares-Barranco, 2013], which is quite different from conventional video cameras. Each pixel generates spikes when its change in brightness reaches a defined threshold; thus, instead of buffering video into frames, the activity of pixels is sent out and processed continuously with time. The level of activity depends on the contrast change; pixels generate spikes faster and more frequently when they are subject to more active change. The sensor is capable of capturing very fast moving objects (e.g., up to 10K rotations per second), which is equivalent to 100K conventional frames per second [Leñero-Bardallo et al., 2011]. However, DVSs on the market are still expensive to purchase, thus we present an extensible behavioural emulator of a DVS using a conventional digital camera, pyDVS [Garibaldi et al., 2016].

The binaural silicon cochlea [Liu et al., 2010] models the elementary functions of the cochlea including the basilar membrane, the Inner Hair Cells (IHCs), and the Spiral Ganglion Cells (SGCs). The input sound wave of each cochlea is filtered by a 64 channel bank of cascaded filters to model the frequency distribution along the basilar membrane. IHCs ~~locating~~ located at each frequency channel perform approximately as half-wave rectifiers, since they release neurotransmitter only when their stereocilia bend in one direction driven by the basilar membrane. The transformation from mechanical waves to electrical action potentials completes at the SGCs, where four pulse-frequency modulators at each frequency channel act like SGCs and generate spikes with individual thresholds.

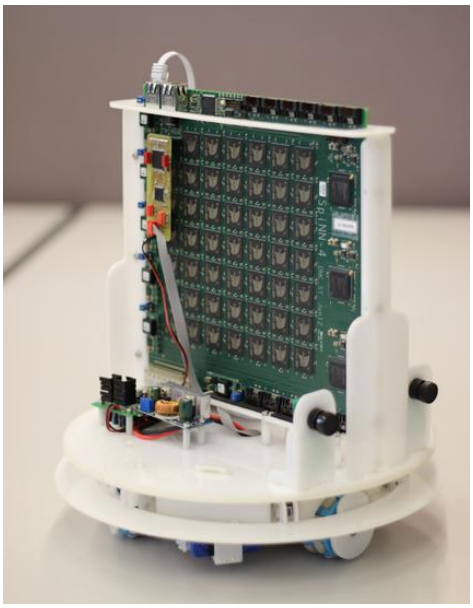
The output spikes from the sensory devices are then communicated to back-end SNN processing using the asynchronous AER protocol. Visual/auditory recognition on such complete neuromorphic hardware systems has emerged in pursuit of efficient energy cost and low sensory latency. However, these hardware SNN simulators either run on FPGAs [Neil and Liu, 2014; Kiselev et al., 2016] or analogue circuits [Qiao et al., 2015] thus are limited in scale.

SpiNNaker provides an ideal platform for real-time visual/auditory processing with

(a) Picture of the neuromorphic visual processing platform. From left to right: a silicon retina, an FPGA board which converts AER packets to SpiNNaker format, and a 48-node SpiNNaker system.



(b) Picture of the neuromorphic auditory processing platform which is similar to (a) in that a silicon cochlea (top) connects to a SpiNNaker board (lower right) through an FPGA (left).



(c) Picture of the omni-directional platform with embedded low-level motor control and elementary sensors: stereo silicon retinas, wheel encoders, and a bump-sensor ring.

Figure 2.9: Three set-ups of neuromorphic sensory and processing hardware platform using SpiNNaker.

large SNN models. Figure 2.9 shows three set-ups of such 'stand-alone' neuromorphic sensory and processing hardware platforms, which can be operated on their own as closed-loop systems (Figure 2.9(c)). The visual system, shown in Figure 2.9(a), ran a 5-layered SNN model [Liu and Furber, 2015] we designed for live gesture recognition, which contained a network of 74,210 neurons and 15,216,512 synapses, and used 290 SpiNNaker cores in parallel and reached 93.0% accuracy. We also successfully implemented a sound localisation model of spiking neurons on an auditory system, see Figure 2.9(b), which could operate with input spikes with sub-millisecond resolution [Lagorce et al., 2015]. These works prove that real-time, large-scale, sensory neuromorphic systems are ready for further study in effective cognition and the genuine intelligence capabilities of such biological-plausible machines.

2.4 Summary

This chapter introduced the structure and behaviour of biological neurons and illustrated how these neural dynamics can be modelled by mathematical abstractions as spiking neurons, which are the basic components of an SNN. The difference between biologically-plausible spiking neuronal operations and rate-based artificial activations holds the key to the research question: how to equip SNNs with cognitive capabilities equivalent to ANNs. Finally, we gave an overview of all the tools and hardware platforms which are used later in the thesis for SNN simulations.

Chapter 3

Deep Learning

Deep Learning research in the field of Artificial Neural Networks (ANNs) has dominated state-of-the-art solutions on cognitive tasks, e.g. the performance exceeding human-level on image classification tasks [He et al., 2015] and playing GO [Silver et al., 2016]. Merging Deep Learning techniques into Spiking Neural Networks (SNNs), introduced in the previous chapter, may provide an answer to the problem of how to operate SNNs to perform equivalently to ANNs in cognitive tasks. In this chapter, we will give an overview of popular architectures and models of Deep Learning in Section 3.1, and the rest of this chapter will illustrate the mechanisms of Convolutional Networks (ConvNets), Autoencoders (AEs), and Restricted Boltzmann Machines (RBMs) in detail which will be used in following chapters.

3.1 Brief Overview

Deep Learning ~~seems to have~~ has become the answer to ~~all~~ increasing number of artificial intelligence problems ~~overnight since~~Geoffrey Hinton since Hinton et al. [2006] firstly proposed the training method of ~~a type of ANN,~~ the Deep Belief Network~~, in 2006 [Hinton et al., 2006]~~. However, Deep Learning is not new 'magic', but rather has a history over a few decades~~and its~~. Its sudden success is the result of the availability of an increasing amount of training data, the growing size of network models and greater computational power. Therefore, all the classical Deep Learning architectures date back to the last century, even before the 'Deep Learning' name was coined. In this section, we briefly introduce some of the influencing techniques.

3.1.1 Classical Models

We call the well-known and widely-used Deep Learning models 'classical' and give a brief introduction to those models in this section. As mentioned above, the first break-through in training deep, as opposed to shallow ($\leq$ 3 layers), networks was the greedy layer-wise strategy [Hinton et al., 2006] proposed to train stacked RBMs, which will be described in more detail in Section 3.4. Shortly after, this method was proved also to be efficient for training other kinds of deep networks including stacked AEs [Bengio et al., 2007] (stated in Section 3.3). RBMs and AEs are suitable for dimensionality reduction and feature extraction when trained with unsupervised learning on unlabelled data. In 2012, using such an unsupervised Deep Learning architecture, the Google Brain team achieved a milestone in the Deep Learning era; the neural network learned to recognise cats by 'watching' 10 million images generated from random frames of YouTube videos [Le, 2013].

ConvNets are biologically inspired from the significant discovery of Hubel and Wiesel that the orientation selectivity (simple cells) and pooling mechanism (complex cells) represent the basic functions in the primary visual cortex in cats [Hubel and Wiesel, 1962]. These simple cells fire at a high frequency to their preferred orientation of visual stimuli within their receptive fields (shown in Figure 2.2), small sub-regions of the visual field. Meanwhile, a complex cell corresponds to the existence of a pattern within a larger receptive field but loses the exact position of the pattern. The NeoCognitron [Fukushima and Miyake, 1982] was the first network to mimic the functions of V1 simple and complex neurons in an ANN, and later this feature detection of single cells was improved by sharing weights among receptive fields in LeNet-5 [LeCun et al., 1998], the typical ConvNet used today. The mechanism of shared weights forms the essence of convolution in a ConvNet, which hugely reduces the number of weight parameters in a network. The most significant milestones produced by deep ConvNet dominated the best performances in the annual ImageNet Challenge [Russakovsky et al., 2015]: AlexNet [Krizhevsky et al., 2012], VGG Net [Simonyan and Zisserman, 2014], GoogLeNet [Szegedy et al., 2015] and ResNet [He et al., 2016].

Despite the powerful capabilities of these feed-forward deep networks, sequence processing is a challenge for them since the size of the input and output vectors are constrained to the number of neurons. Thus Recurrent Neural Networks (RNNs),

containing feed-back connections, are ideal solutions for dealing with sequential information since their current output is always dependent on the previous 'memory'. As training mechanisms have become more mature, for example using the Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], RNNs have shown great success in many natural language processing tasks: language modelling [Mikolov et al., 2010], machine translation [Sutskever et al., 2014], speech recognition [Graves and Jaitly, 2014] and image caption generation [Karpathy and Fei-Fei, 2015].

3.1.2 Combined Approaches

The current trend in Deep Learning is to combine machine learning algorithms towards more complex objectives such as sequential decision making and data generation.

Reinforcement Learning (RL) is inspired from animal behaviour when agents learn to make sequential optimised decisions to control an environment. To address complex decision making problems in practical life, RL requires a sufficiently abstract representation of the high-dimensional environment. Fortunately, Deep Learning just complements the requirement which performs effectively at dimensionality reduction and feature extraction. The milestone achieved by the integration of RL and deep networks, deep RL, drew everyone's attention to artificial intelligence when AlphaGo beat a professional human player at Go [Silver et al., 2016].

Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] are proposed for training generative models of complex data. Instead of training discrimination networks (e.g. image classification using CovNets) and generation networks (e.g. data sampling on RBMs) separately with different objectives, GANs train two competing networks, one the discriminator, the other the generator, simultaneously by continuously making them play games with each other. Thus, the generator learns to produce more realistic data to fool the discriminator; meanwhile the discriminator learns to become better at distinguishing generated from real data. Exciting achievements have been reported in generating complex data such as realistic image generation based on descriptions in text [Radford et al., 2015].

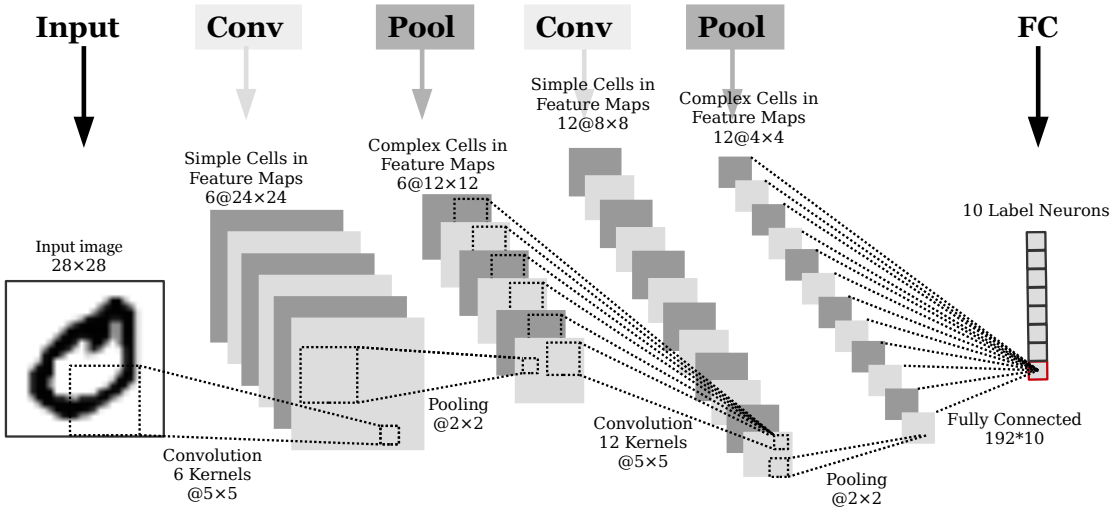


Figure 3.1: Typical ConvNet architecture.

3.2 Convolutional Networks

In Chapter 4, we will use a convolutional network to demonstrate a training method for spiking deep architectures. This section aims to give a detailed introduction to ConvNet architectures and their training with Backpropagation.

3.2.1 Network Architecture

A typical ConvNet, LeNet 3.1, shown in Figure, consists of four types of neural layers: an input layer, convolutional (Conv) layers of simple cells, pooling (Pool) layers of complex cells and fully-connected (FC) layers. From the left to right of Figure 3.1, we illustrate the mechanism of the ConvNet in detail.

The pixels of the input image are normalised and fed into the first Conv layer. Every simple cell takes inputs within its receptive field and the weights on each input pixel are determined by the convolutional kernel which has the same size as the receptive field. The neurons work as demonstrated in Figure 2.6(a) performing a dot product between the weights and the input volume of a receptive field; this is followed by an activation function. Convolving a whole input image with a kernel composes a feature map in the Conv layer, thus in this LeNet example there are 6 feature maps in the first Conv layer. Stride controls the output volume of a convolution such that setting the stride to *step* causes the kernel to move *step* pixels at a time. Thus a small stride produces more output volume and highly overlapping receptive fields. Suppose an

input image and a kernel are both squares, with side lengths of l_{in} and l_k, then the side length of the convolved feature map is $(l_{in} - l_k + 1)/stride$.

The special characteristics of Conv layers lie in the local connectivity and the parameter sharing, such that a neuron connects only to a spatially local input volume, its receptive field; and the weights (the convolutional kernel) are shared among all the receptive fields. Consequently, the number of parameters (trainable weights) hugely decreases compared to all-to-all connections of the same network size.

The complex neurons of the Pool layers either output the maximum input within their receptive fields (max pooling) or the average (average pooling), by applying a max/average filter to non-overlapping receptive fields. The pooling process reduces the spatial size of the feature maps but keeps the number of features. In Chapter 4, we will use average pooling of 2×2 for all the Pool layers, as shown in Figure 3.1. The average filter traverses an entire feature map with a stride of 2, and outputs the averaged element of each receptive field to the next layer.

The next Conv layer drives 3D feature vectors with a depth of 6 (six feature maps) to convolve with 12 3D kernels with size of $6 \times 5 \times 5$. The example demonstrates a common set-up using all the feature maps of the 3D feature vectors; usually it is feasible to select a subset of the input feature maps to involve in the convolutions with each kernel. Then a Pool layer follows the convolution. Repeating these alternating presentations of a Conv layer and a Pool layer builds up a deeper ConvNet.

The trainable shared weights used in the Conv layer hugely reduce the number of weight parameters in a ConvNet, while Pool layers use static convolutional kernels to shrink the size of the network. These convolutional connections can be described as 6c5-2s-12c5-2s, where $\alpha c \beta$ indicates α kernels of side length β used in the Conv layer, and $2s$ specifies the side length and the stride of a pooling kernel. In a ConvNet an FC layer is usually located at the last layer (right in Figure 3.1), and connects the last Pool layer to the output neurons with all-to-all connections. The strongest response among the output neurons indicates to which class the input image belongs, as illustrated in Figure 3.1 where the red neuron represents the first digit out of ten, '0'. There can be more than one FC layer at the end of a ConvNet, thereby building a Multilayer Perceptron (MLP) network. However, in this thesis, we will use only a single FC layer.

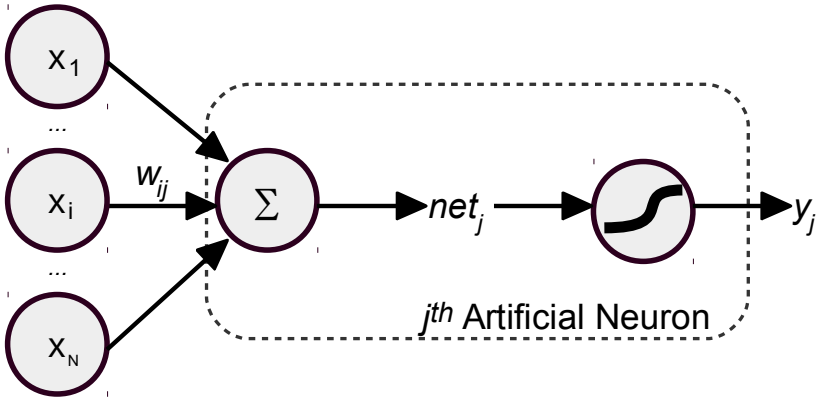


Figure 3.2: N neurons connect to the j_{th} neuron of the next layer. As illustrated in Figure 2.6(a), we use a simplified artificial neuron without bias.

3.2.2 Backpropagation

We have described the feed-forward path of a ConvNet to classify an input image. In this section we will demonstrate the training of a network by back-propagating errors to tune the connection weights. The objective function (or loss function) estimates an error by comparing the output vector of M dimension, $\mathbf{y} = (y_1, y_2, ..., y_M)$, to the desired label vector, $\mathbf{t} = (t_1, t_2, ..., t_M)$, and the error is to be minimised during training. Given a set of training data of K elements $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^K\}$ with desired labels $\mathbf{T} = \{\mathbf{t}^1, \mathbf{t}^2, ..., \mathbf{t}^K\}$ and the network output $\mathbf{Y} = \{\mathbf{y}^1, \mathbf{y}^2, ..., \mathbf{y}^K\}$ the Mean Squared Error (MSE) can be seen as the objective function:

$$L = MSE(\mathbf{T}, \mathbf{Y}) = \frac{1}{2K} \sum_{k=1}^{K} \sum_{m=1}^{M} (y_m^k - t_m^k)^2 \quad . \tag{3.1}$$

~~BP~~ Backpropagation (BP) propagates the gradient of the loss with respect to the weights backwards to each connection in the network. The computation requires a closer look into the structure of a neuron, see Figure 3.2, where N neurons connect to the j^{th} neuron of the next layer, and the neuron converts the weighted summation of the input, net_j, to the output y_j according to its activation function.

Thus the gradient of the loss L with respect to a weight w_{ij} is as follows:

$$\begin{aligned} \frac{\partial L}{\partial w_{ij}} &= \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \delta_j x_i, \quad \text{where} \\ \delta_j &= \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial net_j} \quad , \quad \text{and} \\ x_i &= \frac{\partial net_j}{\partial w_{ij}} \quad . \end{aligned} \tag{3.2}$$

The term δ_j represents the error gradient with respect to net_j, and can be expressed by a recursive definition:

$$\delta_j = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial net_j} = \begin{cases} \frac{1}{K}\sum_{k=1}^{K}(y_j^k - t_j^k)f'(net_j) \ , & \text{if } j \text{ is in an output layer} \\ (\sum_{l}^{L}\delta_l w_{lj})f'(net_j) \ , & \text{otherwise} \end{cases} \tag{3.3}$$

$$, \text{ where } \frac{\partial y_j}{\partial net_j} = f'(net_j) \ , \text{ the derivative of the activation function.}$$

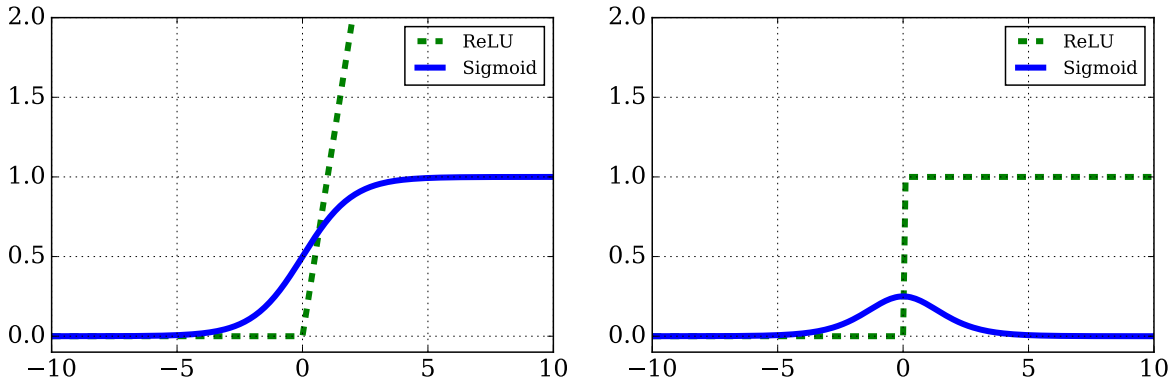
It is easy to obtain the first term, $\frac{\partial L}{\partial y_j}$, of δ_j when j is an output neuron, since it only involves a single dimension of the output vector. However, when j is in an inner layer of the network, which connects to L neurons on the next layer, we have to take the total derivative with respect to y_j: $\sum_l^L \delta_l w_{jl}$. The error propagation applies to any form of connection in a network, such as FC and Conv layers, and the difference only appears in the summation where the matrix product is used for the FC layer and convolution for the Conv layer. In addition, ~~the weights used in BP~~ when BP is implemented, the weights are either transposed in the FC layer or rotated in the Conv layer compared to the forward path.

After error propagation, the BP algorithm updates weights using the optimisation method, gradient descent, to ~~minimize~~ minimise the objective function. It modifies the weights by small steps proportional to the negative of the gradients:

$$\Delta w_{ij} \propto -\frac{\partial L}{\partial w_{ij}} = -\eta \delta_j x_i \ , \tag{3.4}$$

where η defines the length of these updating steps which is also called the learning rate. Again, the weight update is also dependent on the types of layer, where a convolution of the input vector $\mathbf{x}$ with the error gradient δ is needed in Conv layers. A detailed description of BP training on ConvNets can be found elsewhere [Bouvrie, 2006].

Moreover, applying Stochastic Gradient Descent (SGD), the gradient over the full training set can be approximated using only a few, even a single, training data per weight update. Therefore k in Equation 3.3 can be seen as a randomly selected data index, and K represents the number of elements in such a data subset, which is also known as a batch. If we take only one data sample in each batch, the loss function (Equation 3.1) will be estimated by the error between an output vector $\mathbf{y}^k$



(a) Activation functions (b) Derivatives of the activation functions

Figure 3.3: Activation functions: (a) sigmoid and ReLU, and (b) their derivatives.

and the single label sample $\mathbf{t}^k$:

$$E^k = \frac{1}{2}\sum_{m=1}^{M}(y_m^k - t_m^k)^2 \quad , \tag{3.5}$$

and the equation can be simplified by deleting the index k, assuming the error is computed for each data sample:

$$E = \frac{1}{2}\sum_{m=1}^{M}(y_m - t_m)^2 \quad . \tag{3.6}$$

3.2.3 Activation Function and Vanishing Gradient

Each error gradient propagated through layers of the network consists of a chain of multiplied factors as illustrated in Equation 3.3. The deeper a network is and the closer a layer is to the input, the more elements are involved in the multiplication including the derivative of the activation function. Thus, the gradient may vanish if the derivative of the activation function is always less than 1. For the sigmoid function example, shown in Figure 3.3(a), the derivative only peaks to around 0.25 when the input is close to 0, and decays to its minimum as the input either increases or decreases, see Figure 3.3(b).

The Rectified Linear Unit (ReLU) is proposed to tackle the problem of vanishing gradients in deep networks [Nair and Hinton, 2010]. ReLU, simply defined as $y = max(0, x)$ and shown in Figure 3.3, has a gradient of 1 when the input is greater than 0. Hence, the error gradient is able to be back-propagated in deeper networks by multiplying ReLU derivatives.

3.3 Autoencoders (AEs)

Autoencoders ~~(AE)~~ are categorised as unsupervised learning systems, since AE aims to reconstruct its original input as closely as possible. Thus the output and input vectors have the same dimensionality, but the hidden middle layer is allowed to have either more, or fewer, neurons for the purpose of compressing the inputs or mapping them to higher dimensions. Hence the AE is suitable for learning an effective representation of the original inputs at the hidden layer, which can be seen as the encoding phase of the AE, and reconstructing them on the output layer where the hidden vectors are decoded.

Multiple AEs can be stacked to build deep AEs where the hidden layer of a lower AE represents the input of the upper block, and each AE block can be trained individually with greedy layer-wise training [Hinton et al., 2006]. A stacked AE takes advantage of greater expressive power as does any deep network. Taking image processing as an example, the lower AE blocks learn only simple features, such as edges and dots, however on the upper layers more complicated and abstract features are extracted: contours, symbols and even objects.

In this section, we will only describe the structure and training method of single AE blocks, since the greedy layer-wise training on stacked AEs is no different from training multiple single AEs except for different input data. Due to the AE's simple network architecture and training algorithm, spiking AEs will be built and trained with a biologically-plausible training mechanism in Chapter 5.

3.3.1 Structure

Figure 3.4 shows the architecture of an AE, which is an MLP consisting of three layers of neurons: the visible ($\mathbf{v}$), hidden ($\mathbf{h}$) and reconstruction ($\mathbf{v}'$) layers. Both visible and reconstruction layers have N dimensions, and the hidden layer has M. Note that AEs usually have tied weights where the connections ($\mathbf{w}^T$) between $\mathbf{v}$ and $\mathbf{h}$ are transposed to form the connections ($\mathbf{w}$) from $\mathbf{h}$ to $\mathbf{v}'$.

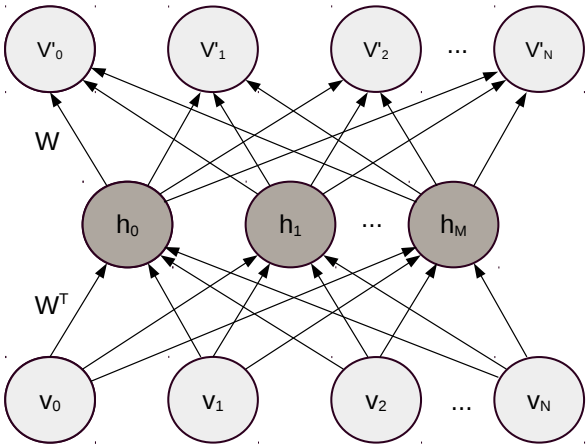


Figure 3.4: A typical Autoencoder structure.

3.3.2 Training

The feedforward path of an AE, as for an MLP, non-linearly transforms the input through two FC layers, where each neuron performs weighted summation and activation as illustrated in Figure 3.2. Training an AE is also similar to that for an MLP, however, the difference lies in the objective where the AE aims to minimise the difference between the input $\mathbf{v}$ and its reconstruction $\mathbf{v}'$, instead of between the output and some labelled data (~~supervised~~ unsupervised learning). Therefore the loss function generated by a single data sample $\mathbf{v}$, described in Equation 3.6 for supervised learning, is as follow:

$$E = \frac{1}{2}\sum_{n=1}^{N}(v'_n - v_n)^2 \quad , \tag{3.7}$$

and the weight update, shown in Equation 3.4, is proportional to the error gradient with respect to the weight:

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}} = -\eta \delta_j h_i \quad , \tag{3.8}$$

where η is the learning rate, h_i is the hidden neuron and the term δ_j is the error gradient with the respect to net_j, the weighted summation. The weight tuning needs only to be applied in the reconstruction layer, since study [Xu, 1993] has proved that updating the transposed weights $\mathbf{w}^T$ at the hidden layer does not improve the learning and the error gradients are usually small. Hence, δ_j can be calculated according to Equation 3.3:

$$\delta_j = (v'_j - v_j) f'(net_j) \quad , \tag{3.9}$$

and the weight update can be simply described by:

$$\Delta w_{ij} = \eta h_i (v_j - v'_j) f'(net_j) \quad . \tag{3.10}$$

If we use ReLU as the activation function, then the above equation is as follow:

$$\Delta w_{ij} = \begin{cases} \eta h_i (v_j - v'_j) \ , & net_j > 0 \\ 0 \ , & net_j <= 0 \end{cases} \tag{3.11}$$

3.4 Restricted Boltzmann Machines (RBMs)

RBMs share similar ideas of unsupervised learning with AEs, but aim to present a data distribution of some training dataset, rather than reconstructing each sample of the dataset. Thus, the RBM is an energy-based model and uses a stochastic approach. Besides that, the structure of an RBM, see Figure 3.5, is also similar to an AE where the visible units **v** bidirectionally connect to the hidden units **h** with strength **w**.

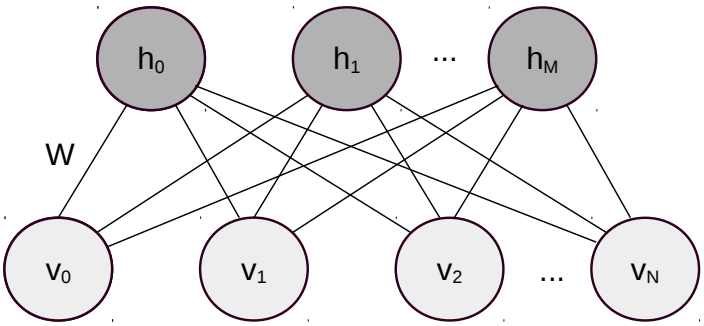


Figure 3.5: A typical RBM structure.

Despite ~~the various statistical models involved in RBMs and their training method~~there are several statistical models and algorithms needed to explain how RBMs work, the actual training ~~algorithm~~ method is rather simple thanks to the use of Contrastive Divergence (CD) ~~[Hinton, 2002]~~, proposed by Hinton [2002]. We can only illustrate some of the statistical models and sampling methods involved in RBM training, a detailed description can be found elsewhere [Fischer and Igel, 2012]. Stacked RBMs are also trained by layer-wise greedy algorithms, thus this section will focus on training a single RBM block.

3.4.1 Energy-based Model

In energy-based models, the probability of data point x is defined by a model function $f(x)$, its energy function $E(x)$ and a partition function Z which normalises the model function to possibilities by adding up all possible $f(x)$:

$$p(x) = \frac{f(x)}{Z} \quad , \quad \text{where} \quad f(x) = e^{-E(x)} \quad , \quad \text{and} \quad Z = \sum_{x} e^{-E(x)} \quad . \tag{3.12}$$

The energy function [Hopfield, 1982] of an RBM is defined as follows:

$$E(\mathbf{v}, \mathbf{h} \mid \Theta) = -\sum_{i=1}^{N} a_i v_i - \sum_{j=1}^{M} b_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i w_{ij} h_j \quad , \tag{3.13}$$

where the visible layer has N units, the hidden layer has M, and Θ are the model parameters used in RBMs $\Theta = \{\mathbf{a}, \mathbf{b}, \mathbf{w}\}$ including biases $\mathbf{a}$ and $\mathbf{b}$ on the visible and the hidden layer respectively, and the weights $\mathbf{w}$ between the layers. As mentioned in Section 2.2, for simplicity we use neurons without biases in this thesis. Therefore only the third term of the complete energy function (Equation 3.13) is kept:

$$E(\mathbf{v}, \mathbf{h} \mid \Theta) = -\sum_{i=1}^{N}\sum_{j=1}^{M} v_i w_{ij} h_j \quad , \tag{3.14}$$

and the model parameters are reduced to $\Theta = \{\mathbf{w}\}$. Thus the joint probability of the visible vector (input) $\mathbf{v}$ and the output of the hidden units $\mathbf{h}$ is defined as follows:

$$\begin{aligned} p(\mathbf{v}, \mathbf{h} \mid \Theta) &= \frac{f(\mathbf{v}, \mathbf{h} \mid \Theta)}{Z(\Theta)} \quad , \text{ where} \\ f(\mathbf{v}, \mathbf{h} \mid \Theta) &= e^{-E(\mathbf{v}, \mathbf{h} \mid \Theta)} \quad , \text{ and} \\ Z(\Theta) &= \sum_{\mathbf{v}} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h} \mid \Theta)} \quad . \end{aligned} \tag{3.15}$$

3.4.2 Objective Function

Given a set of data $\mathbf{D} = \{\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^K\}$, the likelihood function of model parameters Θ defines the probability of the observed outcomes $\mathbf{D}$ given Θ:

$$L(\Theta \mid \mathbf{D}) = p(\mathbf{D} \mid \Theta) = \prod_{k=1}^{K} p(\mathbf{v}^k \mid \Theta) = \prod_{k=1}^{K} \frac{f(\mathbf{v}^k \mid \Theta)}{Z(\Theta)} \quad . \tag{3.16}$$

Thus the objective is to maximise the likelihood $L(\Theta \mid \mathbf{D})$; that is to say, the model parameters Θ that best define the given data $\mathbf{D}$. Furthermore, in order to simplify

the product on the right-hand term in Equation 3.16 the objective function can be replaced with the average log-likelihood:

$$\hat{l}(\Theta \mid \mathbf{D}) = \frac{1}{K} \log L(\Theta \mid \mathbf{D}) = \frac{1}{K} \sum_{k=1}^{K} \log f(\mathbf{v}^k \mid \Theta) - \log Z(\Theta) \quad . \tag{3.17}$$

The derivative of the log-likelihood with respect to a parameter θ is as follows:

$$\begin{aligned} \frac{\partial \hat{l}(\Theta \mid \mathbf{D})}{\partial \theta} &= \frac{1}{K} \frac{\partial \sum_{k=1}^{K} \log f(\mathbf{v}^k \mid \Theta)}{\partial \theta} - \frac{\partial \log Z(\Theta)}{\partial \theta} \\ &= \frac{1}{K} \sum_{k=1}^{K} \frac{\partial \log f(\mathbf{v}^k \mid \Theta)}{\partial \theta} - \sum_{x} p(\mathbf{x} \mid \Theta) \frac{\partial \log f(\mathbf{x} \mid \Theta)}{\partial \theta} \\ &= \left\langle \frac{\partial \log f(\mathbf{v} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{D}} - \left\langle \frac{\partial \log f(\mathbf{v} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{C} \sim p(\mathbf{v} \mid \Theta)} \quad , \end{aligned} \tag{3.18}$$

where $< \cdot >_{\mathbf{X}}$ denotes the mean expectation of $\cdot$ given data set $\mathbf{X}$. The first term of the right-hand side is easy to get with the given data, $\mathbf{d} \in \mathbf{D}$, and the second term can be approximated by generating data samples $\mathbf{C}$ according to $p(\mathbf{x} \mid \Theta)$. The detailed derivation process can be found in the Appendix (Equation A.1). Applying the RBM model function (Equation 3.15) to the Equation 3.18, we derive the simplified representation of the derivative of the log likelihood:

$$\frac{\partial \hat{l}(\Theta \mid \mathbf{D})}{\partial \theta} = \left\langle \frac{\partial - E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\{\mathbf{D}, \mathbf{D}_h \sim p(\mathbf{h} \mid \mathbf{v}, \Theta)\}} - \left\langle \frac{\partial - E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\{\mathbf{C_v}, \mathbf{C_h}\} \sim p(\mathbf{v}, \mathbf{h} \mid \Theta)} , \tag{3.19}$$

where the first term of the right-hand side can be gathered from the given data, $\{\mathbf{D}, \mathbf{D}_h\}$, and the second term will be approximated by sampling $\{\mathbf{C_v}, \mathbf{C_h}\}$. Equation A.2 in the Appendix derives the loss function of RBMs in detail. Then we plug the RBM energy function (Equation 3.14) into the equation above, the loss derivative with respect to the weight is:

$$\frac{\partial \hat{l}(\Theta \mid \mathbf{D})}{\partial w_{ij}} = \langle v_i h_j \rangle_{\{\mathbf{D}, \mathbf{D}_h \sim p(\mathbf{h} \mid \mathbf{v}, \Theta)\}} - \langle v_i h_j \rangle_{\{\mathbf{C_v}, \mathbf{C_h}\} \sim p(\mathbf{v}, \mathbf{h} \mid \Theta)} \quad . \tag{3.20}$$

3.4.3 Contrastive Divergence

Generating data samples for the negative term of Equation 3.20 requires Gibbs sampling on a Markov chain with an infinite number of steps to convergence. Gibbs sampling approximates the joint probability $p(\mathbf{v}, \mathbf{h} \mid \Theta)$ with the conditionally probability defined in RBMs:

$$p(h_j = 1 \mid \mathbf{v}) = \sigma(\sum_{i=1}^{M} w_{ij} v_i) \quad , \quad p(v_i = 1 \mid \mathbf{h}) = \sigma(\sum_{j=1}^{N} w_{ji} h_j) \quad , \tag{3.21}$$

where σ represents the Sigmoid activation function for Bernoulli-Bernoulli RBMs. Figure 3.6 illustrates Gibbs sampling on an RBM by k steps where each pair $(\mathbf{v}, \mathbf{h})$ composes a state of the Markov chain, and the joint probability $p(\mathbf{v}, \mathbf{h} \mid \Theta)$ is approximated by conditional probabilities embedded in the weights (Equation 3.21).

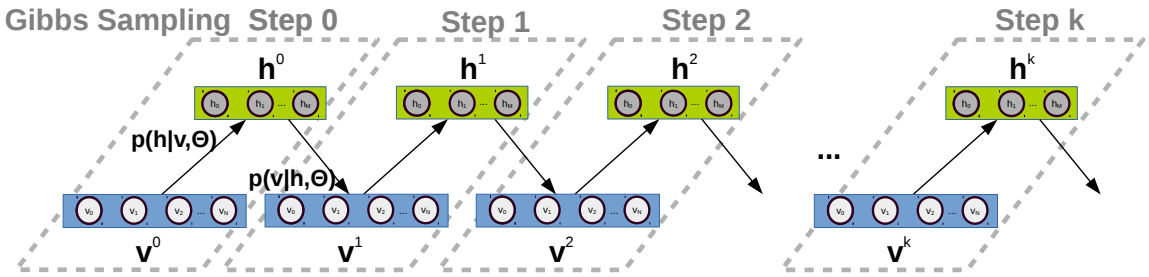


Figure 3.6: Gibbs sampling on a RBM. The method approximates the joint probability $p(\mathbf{v}, \mathbf{h} \mid \Theta)$ by sampling from conditional probabilities $p(\mathbf{h} \mid \mathbf{v}, \Theta)$ and $p(\mathbf{v} \mid \mathbf{h}, \Theta)$. Each sampled data is a pair of $\mathbf{v}^k$ and $\mathbf{h}^k$.

If $k \to \infty$, the Markov chain will converge to an equilibrium which represents the distribution of the model-generated data $\{\mathbf{C_v}, \mathbf{C_h}\}$ described by the RBM, and Equation 3.19 will represent the derivative of the Kullback-Leibler (KL) divergence with respect to parameter θ. The KL divergence measures the distance between two probability distributions: the training dataset $\{\mathbf{D}, \mathbf{D_h}\}$ and the generated Gibbs samples, $\{\mathbf{C_v}, \mathbf{C_h}\}$. However, if we take just a few steps, the KL divergence can be seen as a k-step contrastive convergence (CD_k). Even CD_1 performs surprisingly well in RBM training [Hinton, 2002], which means for every weight update there is only one sample pair (v^1, h^1) generated from Gibbs sampling. Hence, replacing the mean expectation with 1-step Gibbs sampled data in Equation 3.20, the RBM training using SGD can be written as follows:

$$\Delta w_{ij} = \eta(v_i^0 h_j^0 - v_i^1 h_j^1) \quad , \tag{3.22}$$

and v_i^0 is a given training data, h_j^0 is produced with the probability $p(\mathbf{h} \mid \mathbf{v_i^0}, \mathbf{w})$, and the model data pair (v_i^1, h_j^1) is generated by 1-step Gibbs sampling.

3.5 Summary

This chapter briefly introduced the most popular Deep Learning techniques and models for potential use in SNNs. We illustrated the structure and training procedure for three

Deep Learning models: ConvNets, AEs and RBMs, which will be applied in spiking deep networks in the following chapters.

Chapter 4

Generalised Off-line SNN Training

In the last chapter, we described the promising Deep Learning research in the field of Artificial Neural Networks (ANNs). An intuitive idea for bringing these Deep Learning techniques to Spiking Neural Networks (SNNs) is either to transform well-tuned deep ANN models into SNNs or to translate numerical calculations of weight modulations into biologically-plausible synaptic learning rules. Based on the former approach, this chapter proposes a generalised method to train SNNs off-line on equivalent ANNs and transfer the tuned weights back to the SNNs, while the next chapter explores the latter idea of converting Deep Learning algorithms directly into on-line synaptic plasticity rules. Both chapters address the thesis question: how to operate and train SNNs to make them as competent as ANNs on Artificial Intelligence (AI) tasks.

4.1 Introduction

There are two significant problems to be solved when training SNNs off-line. First, an accurate activation function is needed to model the neural dynamics of spiking neurons. In this chapter, we propose a novel activation function used in ANNs, Noisy Softplus (NSP), to closely simulate the firing activity of Leaky Integrate-and-Fire (LIF) neurons driven by noisy current influx. The second problem is to map the abstract numerical values of the ANNs to physical variables, current (nA) and firing rate (Hz), in the SNNs. Consequently we introduce the Parametric Activation Function (PAF), $y = p \times f(x)$, which successfully associates physical units with conventional activation functions thus unifies the representations of neurons in ANNs and the ones in

SNNs. Therefore, an SNN can be modelled and trained on an equivalent ANN using conventional training algorithms, such as Backpropagation (BP).

The significance lies in the simplicity and generalisation of the proposed method. SNN training, now, can be simplified to: firstly, estimate ~~parameter p~~ parameters for the PAF using NSP; secondly, use the PAF version of conventional activation functions to train an equivalent ANN; and finally transfer the tuned weights directly into the SNN without any conversion. Regarding the generalisation, it works exactly the same as training ANNs: the same feed-forward network architecture, BP algorithm and activation functions, and uses the most common spiking neurons, standard LIF, that run on most neuromorphic hardware platforms.

Therefore, most importantly, this research provides the Neuromorphic Engineering (NE) community with a simple, but effective and generalised off-line SNN training method which notably simplifies the development of ~~Artificial Intelligence (AI)~~ AI applications on neuromorphic hardware. In turn, it enables ANN users to implement their models on neuromorphic hardware without the knowledge of spiking neurons or programming specific hardware; thereby enabling them to benefit from the advantages of neuromorphic computers: such as real-time processing, low latency, biological realism and energy efficiency. Furthermore, the success of the proposed off-line training method paves the way to energy-efficient AI on neuromorphic machines scaling from mobile devices to huge computer clusters.

In this chapter, we begin by introducing existing solutions to off-line SNN training in Section ~~4.1~~4.2. We then propose the novel activation function, NSP, and demonstrate how it fits the network dynamics composed of spiking neurons in Section 4.3. Section 4.4 illustrates the generalised SNN training method using PAFs. In Section 4.5 we demonstrate the training of a spiking Convolutional Network (ConvNet), and compare the proposed method to existing training algorithms.

4.2 Related Work

We specified the differences between a conventional artificial neuron and a spiking neuron in Section 2.2: a regular artificial neuron comprises a weighted summation of input data, ~~$\sum x_i w_i$~~ $net_j = \sum x_i w_i$, and an activation function, f, applied to the

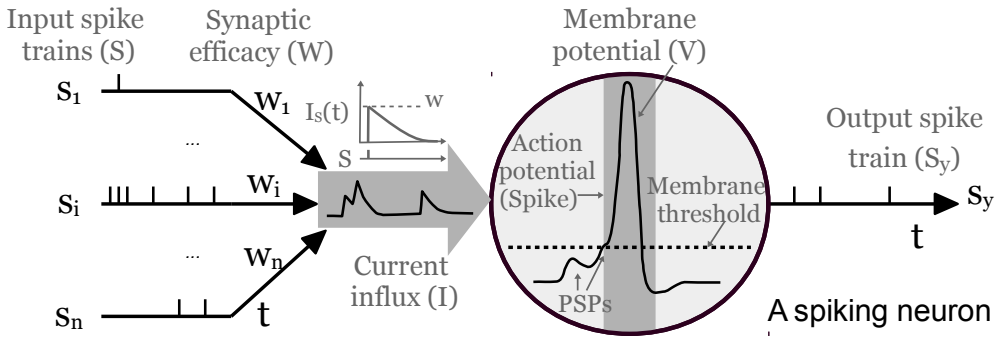


Figure 4.1: A spiking neuron. Spike trains flow into a spiking neuron as current influx, trigger linearly summed PSPs through synapses with different synaptic efficacy **w**, and the post-synaptic neuron generates output spikes when the membrane potential reaches some threshold.

sum, net_j. However the inputs of a spiking neuron (Figure 4.1) are spike trains, which generate current influx through neural synapses (connections). A single spike creates a current pulse with an amplitude of w, which is defined as the synaptic efficacy, and the current then decays exponentially with a decay rate determined by the synaptic time constant, τ_{syn}. The current pulses consequently produce post-synaptic potentials (PSPs) on the neuron driving its membrane potential to change over time, and trigger spikes as outcomes when the neuron's membrane potential reaches some threshold. The dynamics of the current influxes, PSPs, membrane potentials, and spike trains are all time dependent, while the neurons of ANNs only cope with abstract numerical values representing spiking rate, without timing information. Therefore, these fundamental differences in input/output representation and neural computation form the main research problem of how to operate and train biologically-plausible SNNs to make them as competent as ANNs in cognitive tasks. In this chapter, we focus on the solutions of off-line training where SNNs are trained on equivalent ANNs and then the tuned weights are transferred to the SNNs.

Jug et al. [Jug et al., 2012] first proposed the Siegert formula [Siegert, 1951] to model the response function of a spiking neuron, which worked as a Sigmoid unit in training spiking Deep Belief Networks. The Siegert formula maps incoming currents driven by Poisson spike trains to the response firing rate of an LIF neuron, similar to the activation functions used in ANNs which transform the summed input to corresponding outcomes. The variables of the response function are in physical units, thus the trained weights can be transferred directly into SNNs. However, the Siegert

formula is inaccurate as it models the current noise as white [Liu and Furber, 2016], $\tau_{syn} \to 0$, which is not feasible in practice. Moreover, the high complexity of the Siegert function and the computation of its derivative to obtain the error gradient cause much longer training times, thus consuming more energy, when compared to regular ANN activation functions, e.g. Sigmoid. We will illustrate these problems in detail in Section 4.3.

A softened version of the response function of LIF neurons has been proposed [Hunsberger and Eliasmith, 2015] and is less computationally expensive than the Siegert function. However, the model ignores the dynamic noise change introduced by input spikes, assuming a static noise level of the current influx. Therefore the training requires additional noise on the response firing rate and on the training data; however, the manually-added noise is far from the actual activity of the network and includes hyper-parameters in the model.

Although the trained weights can be directly used in SNNs since both the above LIF response functions accept and output variables in physical units, they struggle in terms of poor modelling accuracy and high computational complexity. Moreover, they lose the numerical abstraction of firing rates in ANNs, thus, being constrained to SNN training. Meanwhile, other widely-used activation functions in ANNs cannot be transformed to model SNNs. Therefore, the first problem is the accurate modelling of the neural response activity of LIF neurons using abstract activation functions, in the hope of (1) increasing the modelling accuracy, (2) reducing the computation complexity and (3) generalising off-line SNN training to commonly-used ANN activation functions. These activation functions used in ANNs without physical units are called 'abstract' to differ from the response functions of spiking neurons. We select them for LIF modelling because of the simplicity and generalised use for training ANNs. Thus we propose the activation function, NSP [Liu and Furber, 2016], in Section 4.3.3 to address this problem.

Then the second problem is to map the abstract activation functions to physical units used in SNNs: current in nA and firing rates in Hz. In doing so, the neuronal activities of an SNN can be modelled with such scaled activation functions and the trained weights can be transferred into SNNs without conversion. Instead of directly solving this problem, an alternative way is to train an ANN with abstract activation

functions and then modulate the trained weights to fit in SNNs. Researchers [Cao et al., 2015; Diehl et al., 2015b] successfully applied this method on less biologically-realistic and simplified integrate-and-fire (IF) neurons. Nevertheless, these simple IF neurons are usually difficult to implement in analogue circuits, thus they are feasible only on digital neuromorphic hardware, e.g. TrueNorth [Merolla et al., 2014]. Tuning these trained ANN models to adapt to simplified IF neurons is relatively straightforward, so this method sets the state-of-the-art performance. However, this chapter (in Section 4.4) aims to address the second problem of mapping abstract activation functions to the response firing activity of biologically-plausible LIF neurons. Thus, the training can be not only simplified by using conventional simple activation functions, such as Rectified Linear Units (ReLUs), but also the method can be generalised to target standard LIF neurons which are supported by most neuromorphic hardware.

4.3 Siegert: Modelling the Response Function

The response function, in the context of this thesis, indicates the firing rate of a spiking neuron in the presence of input current. In Section 4.3.1, we introduce the first use of the Siegert formula to model the response function of an LIF neuron. Although the Siegert formula enables off-line SNN training, it has several drawbacks, see Section 4.3.2. Therefore, in Section 4.3.3 we propose the first abstract activation function, NSP, to model the LIF response function.

4.3.1 Biological Background

In Section 2.2.2, we have demonstrated the LIF neuron model as follows:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = V_{rest} - V + R_m I(t) \ . \tag{4.1}$$

The membrane potential V changes in response to the input current I, starting at the resting membrane potential V_{rest}, where the membrane time constant is $\tau_m = R_m C_m$, R_m is the membrane resistance and C_m is the membrane capacitance. The central idea in converting spiking neurons to activation units lies in the response function of a neuron model. Given a constant current injection I, the response function, i.e. firing

rate, of the LIF neuron is:

$$\lambda_{out} = \left[t_{ref} - \tau_m \log \left(1 - \frac{V_{th} - V_{rest}}{IR_m} \right) \right]^{-1}, \text{ when } IR_m > V_{th} - V_{rest}, \tag{4.2}$$

otherwise the membrane potential cannot reach the threshold V_{th} and the output firing rate is zero. The absolute refractory period ~~t_{ref}~~ τ_{refrac} is included, during which period synaptic inputs are ignored.

However, in practice, a noisy current generated by the random arrival of spikes, rather than a constant current, flows into the neurons. The noisy current is typically treated as a sum of a deterministic constant term, I_{const}, and a white noise term, I_{noise}. Thus the value of the current is Gaussian distributed with m_I mean and ${s_I}^2$ variance. The white noise is a stochastic process $\xi(t)$ with mean 0 and variance 1, which is delta-correlated, i.e., the process is uncorrelated in time so that a value $\xi(t)$ at time t is totally independent on the value at any other time t'. Therefore, the noisy current can be seen as:

$$I(t) = I_{const}(t) + I_{noise}(t) = m_I + s_I \xi(t) \quad , \tag{4.3}$$

and accordingly, Equation 4.1 becomes:

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \frac{V_{rest} - V}{\tau_m} + \frac{m_I}{C_m} + \frac{s_I \xi(t)}{C_m} \quad . \tag{4.4}$$

We then multiply both sides of Equation 4.4 by a short time step $\mathrm{d}t$, the stochastic differential equation of the membrane potential is:

$$\begin{aligned} \mathrm{d}V &= \frac{V_{rest} - V}{\tau_m}\mathrm{d}t + \frac{m_I}{C_m}\mathrm{d}t + \frac{s_I}{C_m}\mathrm{d}W_t \\ &= \frac{V_{rest} - V}{\tau_m}\mathrm{d}t + \frac{m_I}{C_m}\mathrm{d}t + \frac{s_I\sqrt{\mathrm{d}t}}{C_m} r \\ &= \frac{V_{rest} - V}{\tau_m}\mathrm{d}t + \mu_v \mathrm{d}t + \sigma_v r \quad . \end{aligned} \tag{4.5}$$

The last term $\mathrm{d}W_t$ is a Wiener process; $W_{t+\mathrm{d}t} - W_t$ obeys a Gaussian distribution with mean 0 and variance $\mathrm{d}t$; thus r is a random number sampled in accordance with a Gaussian distribution of zero mean and unit variance. The instantaneous mean ~~μ and variance σ^2~~ μ_v and variance ${\sigma_v}^2$ of the change in membrane potential characterise the statistics of V in a short time range, and they can be derived from the statistics of the noisy current:

$$\mu_v = \frac{m_I}{C_m}, \qquad \sigma_v = \frac{s_I\sqrt{\mathrm{d}t}}{C_m} \quad . \tag{4.6}$$

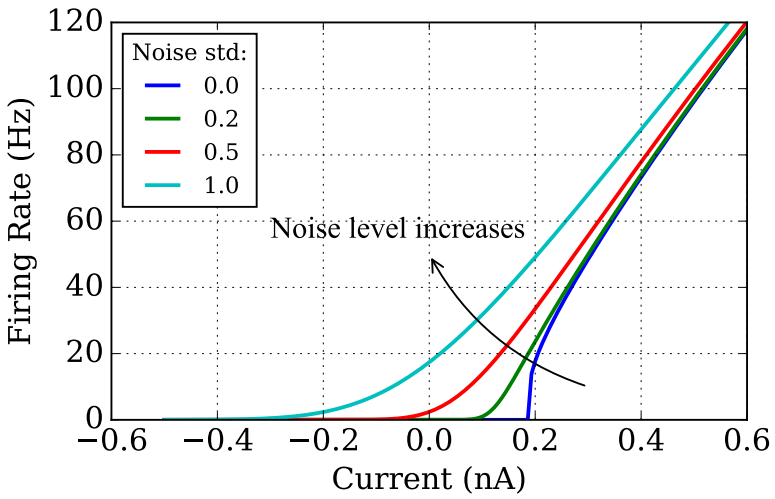


Figure 4.2: Response function of the LIF neuron with noisy input currents with different standard deviations.

The response function [Rauch et al., 2003; La Camera et al., 2008] of the LIF neuron to a noisy current, also known as the Siegert formula, is a function of ~~μ and σ:~~ μ_v and σ_v:

$$\lambda_{out} = \left[t_{ref} + \tau_m \int_{\frac{V_{rest}-\mu\tau_m}{\sigma\sqrt{\tau_m}}}^{\frac{V_{th}-\mu\tau_m}{\sigma\sqrt{\tau_m}}} \sqrt{\pi} \exp(uV^2)(1 + erf(uV))\mathrm{d}uV \right]^{-1} \quad . \quad (4.7)$$

Figure 4.2 shows the response curves (Equation 4.7) of an LIF neuron driven by noisy currents where the Gaussian noise is of mean m_I and standard deviation s_I. The parameters of the LIF neuron are all biologically ~~valid~~ plausible (see the listed values in Table 4.1), and the same parameters are used throughout this chapter. The ~~solid~~ bottom (zero noise) line in Figure 4.2 illustrates the response function of such an LIF neuron injected with constant current, which inspired the proposal of ReLUs. As noise increases, the level of firing rates also rises. Thus the Softplus function approximates the response activity to noisy current, but only represents a single level of noise; for example, the ~~dotted~~ top line in Figure 4.2 shows the curve when $s_I = 1$.

4.3.2 Mismatch of The Siegert Function to Practice

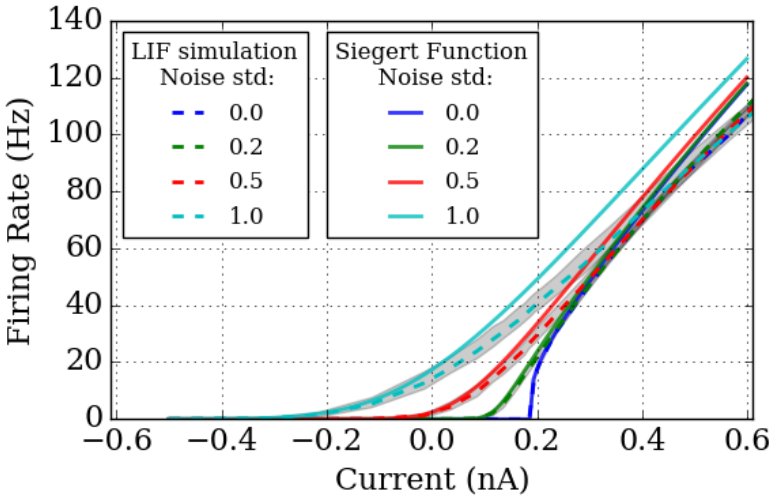
To verify the Siegert function in practice, simulation tests were carried out using PyNN [Davison et al., 2008] to compare the reality with the analytical results (the Siegert function). The noisy current was produced by *NoisyCurrentSource* in PyNN

Table 4.1: Parameter setting for the current-based LIF neurons using PyNN.

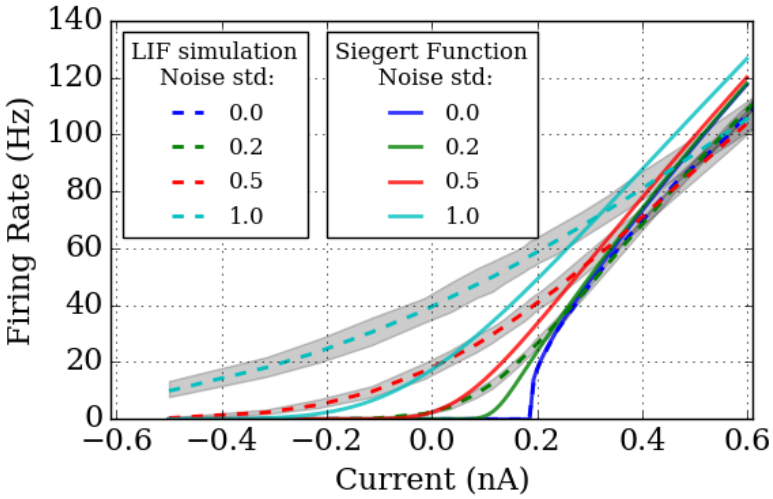
Parameters	Values	~~Units~~ Description
~~cm~~ C_m	0.25 ~~nF~~ nF	membrane capacitance
~~tau_m~~ τ_m	20.0 ~~ms~~ ms	membrane time constant
~~tau_refrac~~ τ_{refrac}	1.0 ~~ms~~ ms	refractory period
~~v_reset~~ V_{reset}	-65.0~~mV~~ mV	resting membrane potential
~~v_rest~~ V_{rest}	-65.0~~mV~~ mV	resetting membrane potential
~~v_thresh~~ V_{thresh}	-50.0~~mV~~ mV	membrane threshold
~~i_offset~~ I_{offset}	~~0.1 nA~~ 0.0 nA	offset of current influx

which is a constant current of m_I added to a Gaussian white noise of zero mean and s_I^2 variance. The noise was drawn from the Gaussian distribution in a time resolution of dt. We chose $dt = 1$ ms which is the finest resolution for common SNN simulators, and $dt = 10$ ms for comparison. For a given pair of m_I and s_I^2, a noisy current was injected into a current-based LIF neuron for 10 s, and the output firing rate was the average over 10 trials. There were four noise levels tested in the experiments: 0, 0.2, 0.5, 1; and the mean current increased from -0.5 to 0.6 nA.

The dashed curves in Figures 4.3 illustrate the output firing rate ~~driven by different levels of noise. The differences relative to the analytical results (thin lines)is due to the~~ of the LIF simulations; while the bold lines are the analytical reference, the Siegert function (same as Figure 4.2). The differences between the practical simulations and the Siegert function enlarge when the time resolution, dt, of the *NoisyCurrentSource* increases from 1 ms (Figure 4.3(a)) to 10 ms (Figure 4.3(b)). The sampled ~~signals~~ current signals (*NoisyCurrentSource*) are shown in Figures 4.4 (a) and (b). The discrete sampling of the noisy current introduces time step correlation to the white noise, shown in Figures 4.4 (e) and (f), where the value remains the same within a time step dt. Although both current signals follow the same Gaussian distribution, see Figures 4.4 (g) and (h), the current is approximately a white noise when $dt = 1$ ms, but a coloured noise, e.g. ~~increased~~ increases Power Spectral Density (PSD) at lower frequency, when $dt = 10$ ms, see Figures 4.4 (c) and (d). ~~Thus the~~ Therefore, the coloured noise of the current influx drives the LIF neuron fire observably more intensely. Hence,

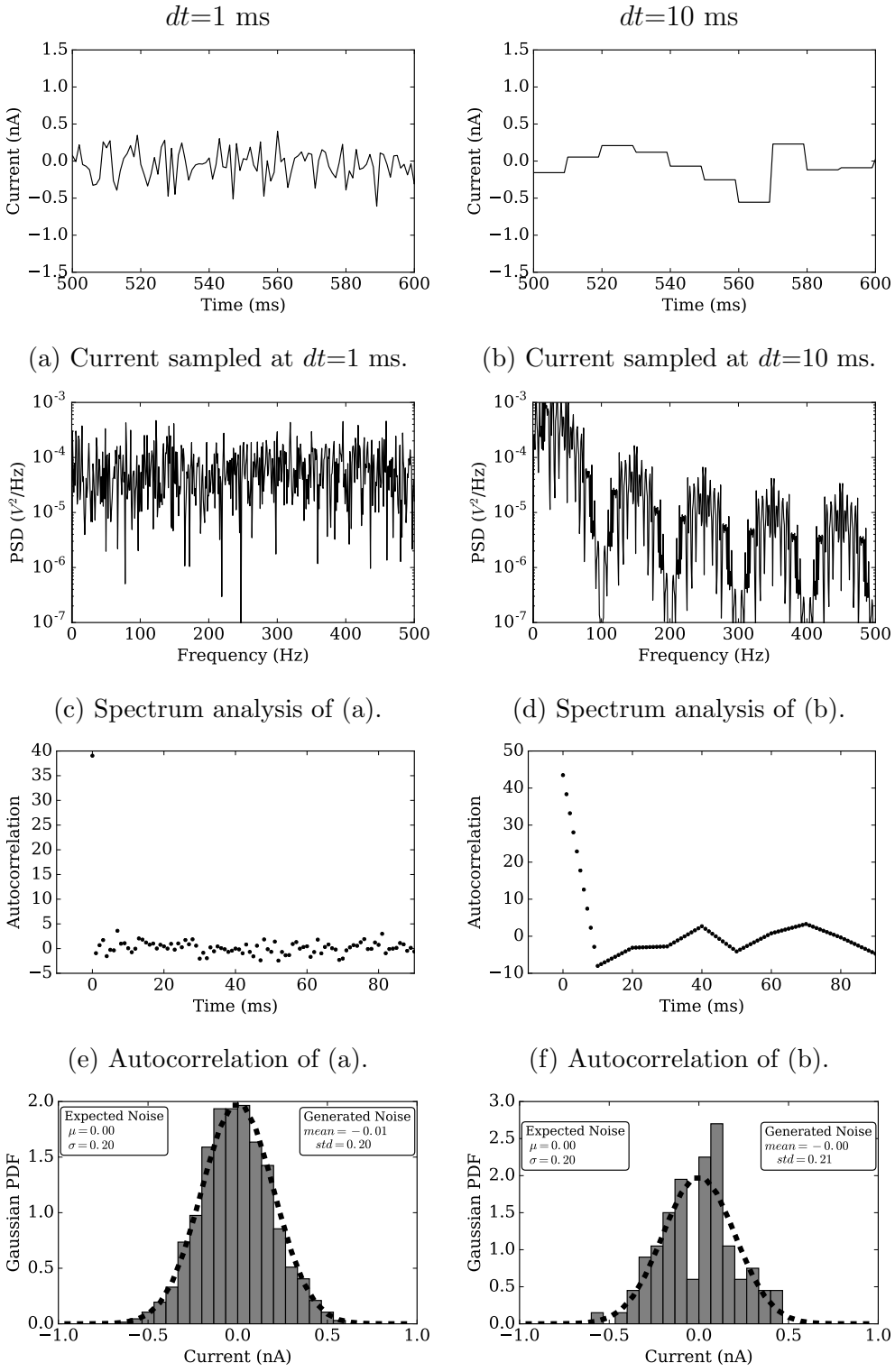


(a) Current sampled at dt=1 ms.



(b) Current sampled at dt=10 ms.

Figure 4.3: Recorded response firing rate of an LIF neuron driven by *NoisyCurrentSource* in PyNN, compared to the Siegert function. The *NoisyCurrentSource* is sampled at every (a) 1 ms and (b) 10 ms. Averaged firing rates of 10 simulation trails tested on four noisy levels are shown in ~~bold~~ different colours of dashed lines, and the grey colour fills the range between the minimum to maximum of the firing rates. The analytical LIF response function, the Siegert formula (Equation 4.7), is drawn in ~~thin~~ bold lines (shown in Figure 4.2) to compare with the practical ~~simulation~~simulations. The selected noise levels are the same with the LIF simulations, and the curves are plotted with the same colours of the dashed lines.



(a) Current sampled at dt=1 ms.

(b) Current sampled at dt=10 ms.

(c) Spectrum analysis of (a).

(d) Spectrum analysis of (b).

(e) Autocorrelation of (a).

(f) Autocorrelation of (b).

(g) Distribution of samples of (a).

(h) Distribution of samples of (b).

Figure 4.4: *NoisyCurrentSource* samples noisy currents from a Gaussian distribution every 1 ms (left) and 10 ms (right). The signals are shown in the time domain in (a) and (b), and in the spectrum domain in (c) and (d). The autocorrelation of both current signals are shown in (e) and (f). The distribution of the discrete samples are plotted in bar charts to compare with the PDF of the original Gaussian distribution, shown in (g) and (h).

the Siegert formula, Equation 4.7, can only approximate the LIF response of noisy current with white noise. ; but is not adapted to coloured noise. In practice, the current is generated by random arrivals of input spikes with various synaptic efficiency, which also brings in coloured noise.

τ_{syn}=1 ms. τ_{syn}=10 ms. Recorded response firing rate of an LIF neuron driven by a synaptic current with two synaptic time constants: (a) τ_{syn}=10 ms and (b) τ_{syn}=10 ms. Average firing rates of 10 simulation trails are shown in bold lines, and the grey colour fills the range between the minimum to maximum of the firing rates. The firing rates recorded driven by *NoisyCurrentSource* are drawn in thin lines (shown in Figure 4.4) for comparison.

τ_{syn}=1 ms τ_{syn}=10 ms Current sampled at dt=1 ms. Current sampled at dt=10 ms. Spectrum analysis of (a). Spectrum analysis of (b). Autocorrelation of (a). Autocorrelation of (b). Distribution of samples of (a). Distribution of samples of (b). Noisy currents generated by 100 Poisson spike trains to an LIF neuron with synaptic time constant τ_{syn}=1 ms (left) and τ_{syn}=10 ms (right). The currents are shown in the time domain in (a) and (b), and in the spectrum domain in (c) and (d). The autocorrelation of both current signals are shown in (e) and (f). The distribution of the generated samples are plotted in bar chart form to compare to the expected Gaussian distribution, shown in (g) and (h).

A more realistic simulation of a noisy current can be generated by 100 Poisson spike trains, where the mean and variance of the current are given by [La Camera et al., 2008]:

$$m_I = \tau_{syn} \sum_i w_i \lambda_i \ , \ s_I^2 = \frac{1}{2} \tau_{syn} \sum_i w_i^2 \lambda_i \ , \tag{4.8}$$

where τ_{syn} is the synaptic time constant, and each Poisson spike train connects to the neuron with a strength of w_i and a firing rate of λ_i. Two populations of Poisson spike sources, for excitatory and inhibitory synapses respectively, were connected to a single LIF neuron to mimic the noisy currents. The firing rates of the Poisson spike generators were determined by the given m_I and s_I. Figure 4.5 illustrates the recorded firing rates responding to the Poissoin spike trains compared to the mean firing rate driven by *NoisyCurrentSource* in Figure 4.3. Note that the estimation of LIF response activity using the Siegert function requires noisy current with white noise, however in practice the release of neurotransmitter takes time ($\tau_{syn} >> 0$) and the synaptic current decays

exponentially $I_{syn} = I_0 e^{-\frac{t}{\tau_{syn}}}$. Figures 4.6 (a) and (b) show two examples of synaptic current of 0 nA mean and 0.2 standard deviation driven by 100 neurons firing at the same rate and with the same synaptic strength (half excitatory, half inhibitory), but of different synaptic time constant. Therefore, the current at any time t during the decay period is dependent on the value at the previous time step, which makes the synaptic current a coloured noise, see Figures 4.6 (c) and (d).

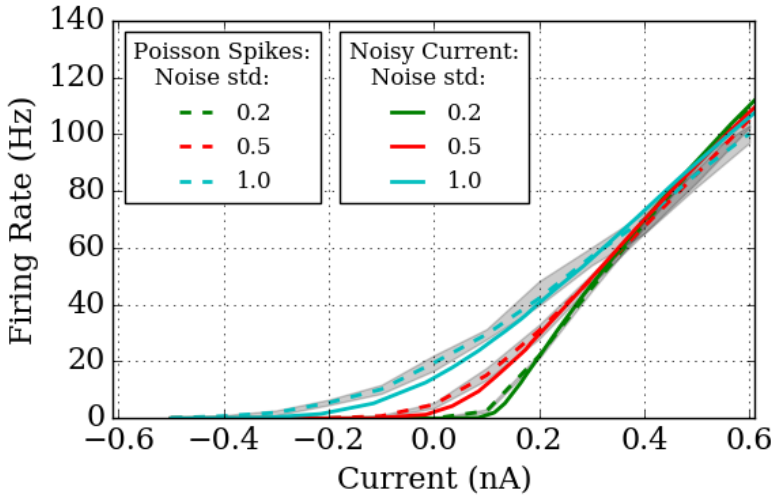
We observe in Figure 4.5 (a) that the response firing rate to synaptic current is higher than the *NoisyCurrentSource* for ~~all the 10 trials~~most of the current range. This is caused by the coarse resolution (1 ms) of the spikes, thus the standard deviation of the current is larger than 0.2, shown in Figure 4.6 (g); and the τ_{syn}, even when as short as 1 ms, adds coloured noise instead of white noise to the current. However, Figure 4.5 (b) shows a similar firing rate of both the synaptic driven current and the *NoisyCurrentSource*, since both of the current signals have similar distribution (Figure 4.6 (h)) and time correlation (Figure 4.6 (f)). Nevertheless, the analytical response function, the Siegert formula, cannot approximate either of the practical simulations (see Figure 4.3).

Although the use of the Siegert function opened the door for modelling the LIF response function to work similarly to the activation functions used in ANNs [Jug et al., 2012], there are several drawbacks of this method:

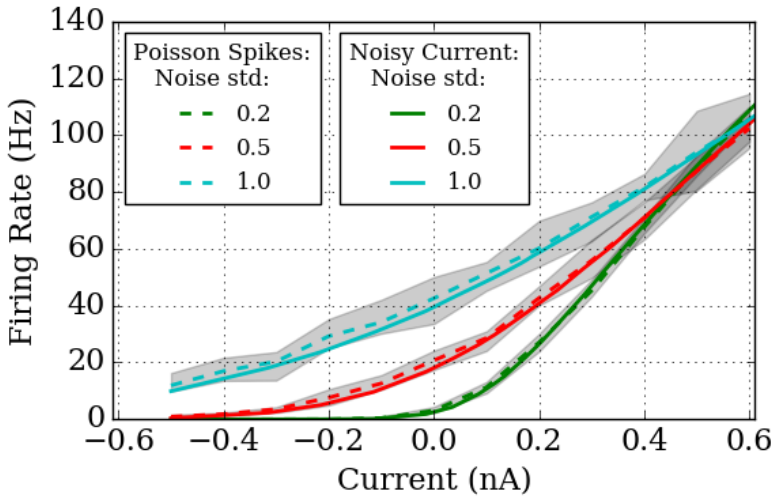
- most importantly, the numerical analysis of an LIF response function is far from accurate in practice. 'Practice' in the chapter means SNN simulations using LIF neurons. Thus the inaccurate model generates errors between the estimation and the practical response firing rate.

- the high complexity of the Siegert function causes much longer training times and more energy, let alone the high-cost computation on its derivative.

- The Siegert function is used to replace sigmoid functions for training spiking Restricted Boltzmann Machines (RBM) [Jug et al., 2012]. Therefore neurons have to fire at high frequency (higher than half of the maximum firing rate) to represent the activation of a sigmoid unit; thus the network activity results in high power dissipation.

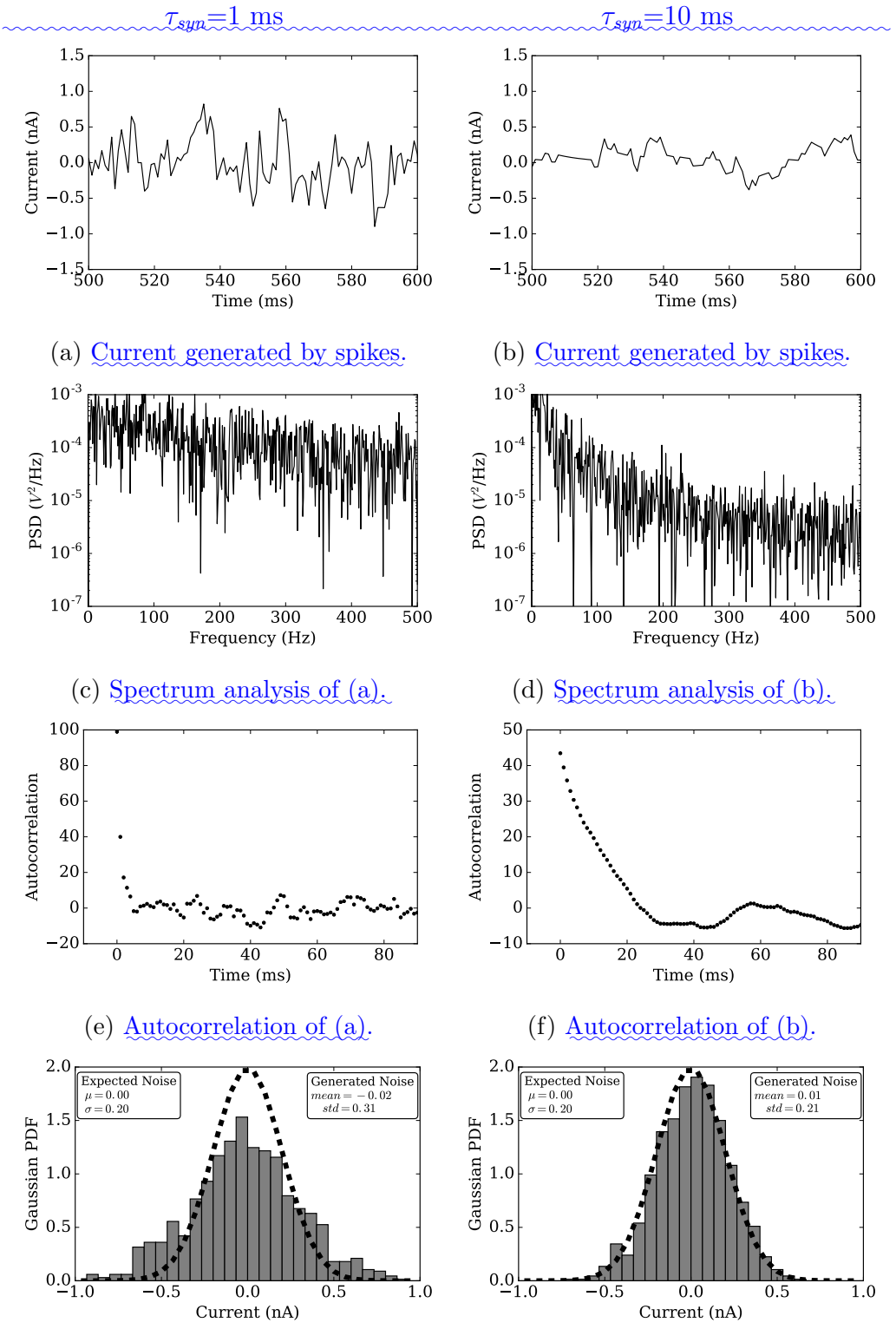


(a) τ_{syn}=1 ms.



(b) τ_{syn}=10 ms.

Figure 4.5: Recorded response firing rate of an LIF neuron driven by a noisy synaptic current, which is generated by random arrivals of Poisson spike trains, compared to previous experiments using *NoisyCurrentSource*. Averaged firing rates of 10 simulation trails tested on three noisy levels are shown in different colours of dashed lines, and the grey colour fills the range between the minimum to maximum of the firing rates. The other LIF simulation using *NoisyCurrentSource* is drawn in bold lines (same as the dashed lines in Figure 4.3) to compare with the noisy synaptic current. The same noise level is plotted with the same colour for both experiments. Two synaptic time constants are tested: (a) τ_{syn}=1 ms, to compare with *NoisyCurrentSource* sampled at every 1 ms, and (b) τ_{syn}=10 ms, to compare with *NoisyCurrentSource* sampled at every 10 ms.



(a) Current generated by spikes. (b) Current generated by spikes.

(c) Spectrum analysis of (a). (d) Spectrum analysis of (b).

(e) Autocorrelation of (a). (f) Autocorrelation of (b).

(g) Distribution of samples of (a). (h) Distribution of samples of (b).

Figure 4.6: Noisy currents generated by 100 Poisson spike trains to an LIF neuron with synaptic time constant τ_{syn}=1 ms (left) and τ_{syn}=10 ms (right). The currents are shown in the time domain in (a) and (b), and in the spectrum domain in (c) and (d). The autocorrelation of both current signals are shown in (e) and (f). The distribution of the generated samples are plotted in bar chart form to compare to the expected Gaussian distribution, shown in (g) and (h).

- better learning performance has been reported using ReLU than Sigmoid units, so modelling spiking neurons with a ReLU-like activation function ~~for spiking neurons~~ is needed.

Therefore, we propose the NSP function which provides solutions to the drawbacks of the Siegert unit.

4.3.3 Noisy Softplus (NSP)

Due to the limited time resolution of common SNN simulators and the time taken for neurotransmitter release, τ_{syn}, mismatches exist between the analytical response function, the Siegert formula, and practical neural activities. Consequently to model the practical LIF response function (see Figure 4.7(a)) whose output firing rates are determined by both the mean and variance of the noisy input currents, the NSP ~~was~~ is proposed as follows:

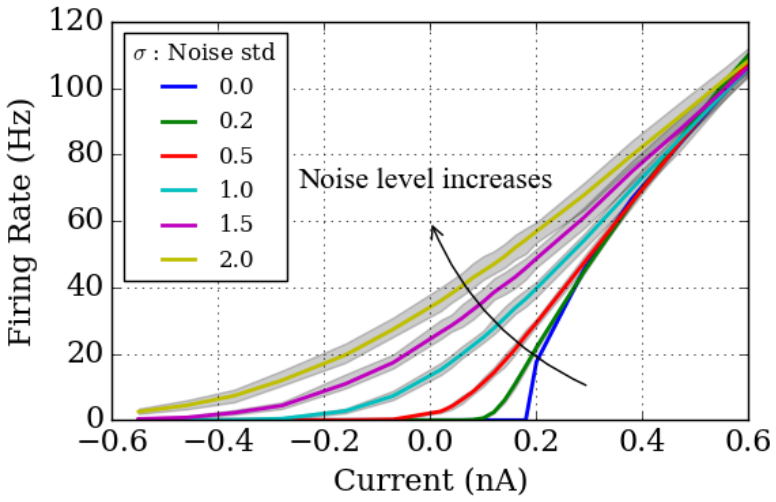
$$y = f_{nsNSP}(x, \sigma) = k\sigma \log[1 + \exp(\frac{x}{k\sigma})] \ , \tag{4.9}$$

where x and σ refer to the mean and standard deviation of the input current, y indicates the intensity of the output firing rate, and k, determined by the biological configurations on the LIF neurons [Liu and Furber, 2016] (listed in Table 4.1), ~~controls the~~ scales the impact of the noise thereby controlling the shape of the curves. Note that the novel activation function we proposed contains two parameters, the mean current and its noise, which ~~can be~~ takes the values estimated by Equation 4.8:

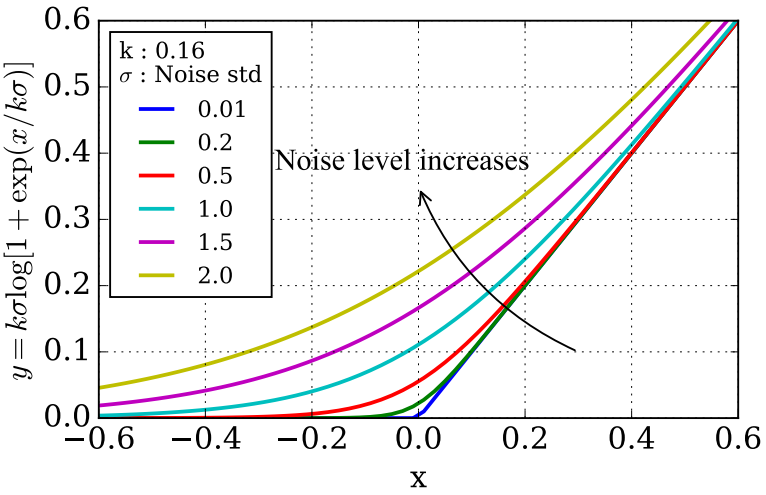
$$x = \tau_{syn} \sum_i w_i \lambda_i \ , \ \sigma^2 = \frac{1}{2} \tau_{syn} \sum_i w_i^2 \lambda_i \ ,$$

~~where λ_i indicates the firing rate of an input spike train~~m_I and s_I^2. Since the NSP takes two variables as inputs, the activation function can be plotted in 3D, see Figure 4.8.

Figure 4.7(b) shows the activation function in curve sets corresponding to different discrete noise levels which mimics the responding activities of practical simulations of LIF neurons, shown in Figure 4.7(a). ~~Since the NSP takes two variables as inputs, the activation function can be plotted in 3D, see Figure 4.8.~~ It is note worthy that, the non-smooth curve (blue line in Figure 4.7(a) generated by $\sigma = 0$) of the LIF response activities does not fit into the NSP function. It is a limitation of using



(a) Response firing rate of an LIF neuron



(b) NSP

Figure 4.7: NSP models the LIF response function. (a) Firing rates measured by simulations of an LIF neuron driven by different input currents and discrete noise levels. Bold lines show the average and the grey colour fills the range between the minimum and the maximum. (b) NSP activates the input x according to different noise levels where the noise scaling factor $k = 0.16$.

NSPs to model spiking rates when the noise level approaches to 0. However, we ignore the minor mismatch to unify and simplify the model, since the results show an acceptable performance drop in Section 4.5. In addition, scaling, shifting and parameter calibrations are essential to accurately fit the NSP to actual LIF response activities. We will illustrate the procedure in Section 4.4.1.

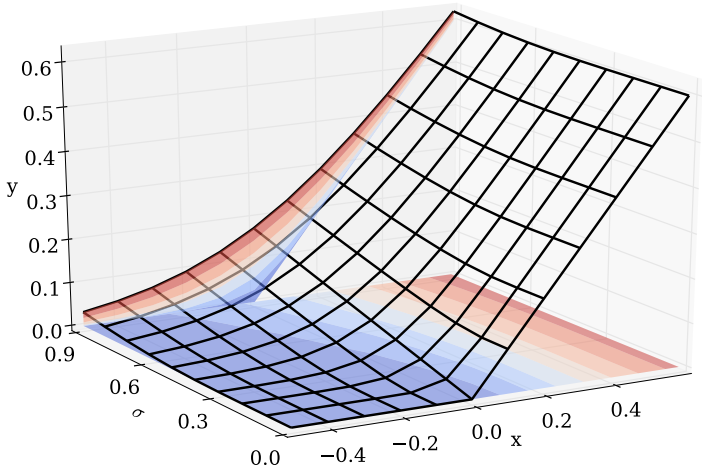


Figure 4.8: Noisy Softplus in 3D.

The derivative of the NSP is the logistic function scaled by $k\sigma$:

$$\frac{\partial f_{ns}(x,\sigma)}{\partial x}\frac{\partial f_{NSP}(x,\sigma)}{\partial x} = \frac{1}{1+exp(-\frac{x}{k\sigma})} \quad , \tag{4.10}$$

which could be applied easily to back propagation in any ANN training.

4.4 Generalised Off-line SNN Training

We have briefly discussed modelling the ~~response firing activity~~ LIF response activities of a LIF neuron with an abstract activation function, NSP. ~~However, before we can replace an LIF neuron with an abstract activation function, it is necessary to~~ Function transformations are essential to precisely map the numerical values of NSP to physical variables in SNNs. Therefore, to optimise the modelling of the LIF response function, we curve fit parameters of the NSP (e.g. Figure 4.7(b)) to approximate the actual firing activities (e.g. Figure 4.7(a)) in Section 4.4.1. Section 4.4.2 includes the parameters in the proposed activation function, PAF, to unify the presentation of activation functions for both ANNs and SNNs. Thus, a generalised off-line SNN training method (Section 4.4.3) is completed using PAF. Moreover, a corresponding fine tuning method is put forward to increase the training capability in Section 4.4.4.

4.4.1 Mapping NSP to Concrete Physical Units

The inputs of the NSP function, x and σ, are obtained from physical variables as stated in Equation ~~??~~4.8: m_I and s_I, thus they are already in physical units (nA).

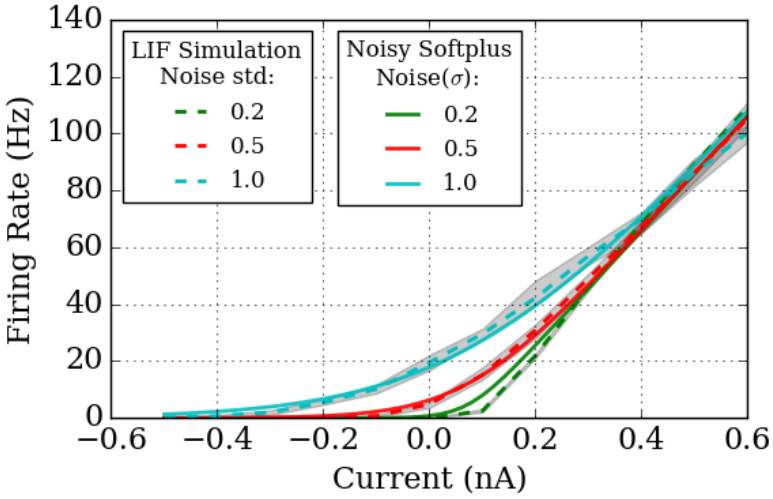
Therefore, linearly scaling up the activation function by a factor S (Hz / nA) can approximate the output firing rate λ_{out} of an LIF neuron in Hz~~:~~. Moreover, including a bias b on the input x allows the curves set move freely on the x-axis to better fit into the actual firing activities:

$$\lambda_{out} \simeq f_{ns\,NSP}(\frac{x}{k\sigma} - \frac{b}{k\sigma}, \sigma) \times S = k\sigma \log[1 + \exp(\frac{x}{k\sigma} - \frac{b}{k\sigma})] \times S \ . \tag{4.11}$$

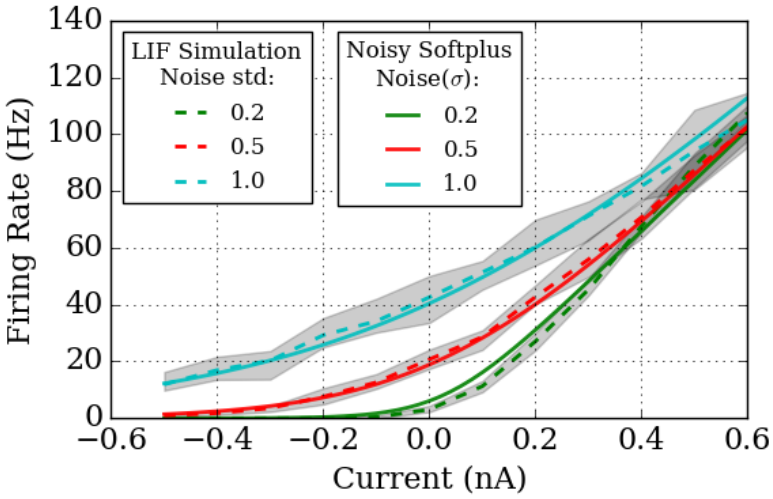
Suitable calibrations of the noise scaling factor k~~and~~, input bias b and mapping scaling factor S in Equation 4.11 enable NSP to closely match the practical response firing rates of LIF neurons given various biological parameters. The ~~parameter pair of (k, S) is~~ parameters of (k, b, S) are curve-fitted with the triple data points of $(\lambda_{out}, x, \sigma)$ and the calibration currently is conducted by linear least squares regression. The output firing rate λ_{out} is measured from SNN simulations where an LIF neuron is driven by synaptic input currents of Poisson spike trains; and x and σ take the mean and variance of the noisy current by using Equation 4.8. Figure 4.9 shows two calibration results in which the parameters were fitted to ~~$(k, S) = (0.19, 208.76)$~~ $(k, b, S) = (0.18, 0.07, 201.66)$ when the synaptic constant is set to $\tau_{syn} = 1$ ms and was fitted to ~~$(k, S) = (0.35, 201.06)$~~ $(k, b, S) = (0.35, 0.03, 178.91)$ when $\tau_{syn} = 10$ ms.

~~τ_{syn}=1 ms τ_{syn}=10 ms NSP fits to the response firing rates of LIF neurons in concrete physical units. Recorded response firing rate of an LIF neuron driven by synaptic current with two synaptic time constants: (a) τ_{syn}=1 ms and (b) τ_{syn}=10 ms. Averaged firing rates of simulation trails are shown in bold lines, and the grey colour fills the range between the minimum to maximum of the firing rates. The thin lines are the scaled NSP.~~ To keep the simple format of traditional activation functions, $y = f(x)$, which has no constant bias on the input, it is easy to pass the bias b to the LIF parameter, the constant current offset, $I_{offset} = b$. Therefore the specific parameter I_{offset} of the LIF neuron is not arbitrarily chosen, but configured by precise estimation on b. More importantly, setting I_{offset} properly on LIF neuron instead of having a constant bias on the input of an activation function keeps the hyper-parameters unchanged in ANN training. For example, the initial weights of a network have to be carefully set to adapt to a constant bias on the activation function.

4.4.2 Parametric Activation Functions (PAFs)



(a) τ_{syn}=1 ms



(b) τ_{syn}=10 ms

Figure 4.9: NSP fits to the response firing rates of LIF neurons in concrete physical units. Averaged firing rates of 10 simulation trails tested on three noisy levels are shown in different colours of dashed lines, and the grey colour fills the range between the minimum to maximum of the firing rates (same as the dashed lines in Figure 4.5). The bold lines are the scaled NSP, where the same noise level is plotted with the same colour as the LIF simulations. The parameters used in the experiments are as follows: (a) τ_{syn}=1 ms for LIF simulation, and k=0.18, S=201.66, b=0.07 for NSP; (b) τ_{syn}=10 ms for LIF simulation, and k=0.35, S=178.91, b=0.03 for NSP.

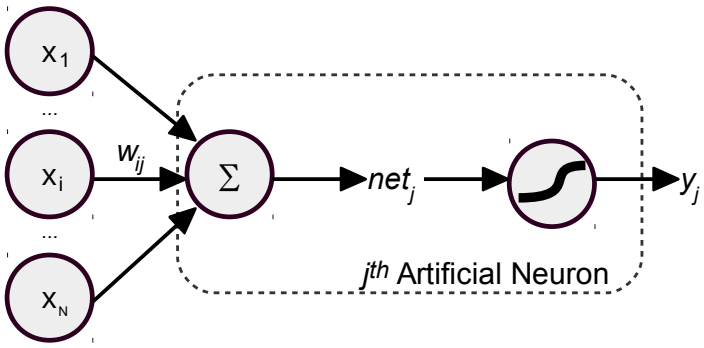


Figure 4.10: A copy of Figure 3.2 showing a general artificial neuron where an activation function transforms the weighted sum net_j to its outcome y_j.

~~An artificial spiking neuron modelled by NSP, where firing rates are the input and output of the neuron. NSP transforms the noisy current influx including both the mean (net_j, solid lines) and the variance σ_j^2 (dashed lines) to the abstract firing rate y_j. An artificial spiking neuron modelled by PAF-NSP, whose input and output are numerical values, equivalent to those of ANNs. PAF includes the scaling factors S and the synaptic time constant τ_{syn} in the combined activation function. A copy of Figure 3.2 showing a general artificial neuron where an activation function transforms the weighted sum net_j to its outcome y_j. Comparisons of three activation functions used in ANNs: (a) the NSP, (b) the PAF-NSP, and (c) a conventional ANN activation function. The PAF links the firing activity of a spiking neuron to the numerical value of ANNs.~~

Neurons in ANNs take inputs from their previous layer, and feed the weighted sum of their input, $net_j = \sum_i w_{ij} x_i$, to the activation function. The transformed signal then forms the output of an artificial neuron, which can be denoted as $y_j = f(net_j)$, see Figure ??~~(c). However, a spiking neuron modelled by NSP takes the firing rate as its input /output, thus Equation ?? can be written as:~~

$$net_j = \sum_i w_{ij}(\lambda_i \tau_{syn}) \ , \ \sigma_j^2 = \sum_i (\frac{1}{2} w_{ij}^2)(\lambda_i \tau_{syn}) \ ,$$

~~and~~ 4.10.

Equation 4.8 illustrates the physical interpretation of the input of an NSP function, the noisy current influx, which has the mean of m_I, and the variance of σ_2. To express the physical parameters with the same form of the weighted summation, net, in a conventional ANN, the mean and variance of the noisy current influx can be

represented with net_x and $net\ \sigma^2$:

$$net\_x_j = \sum_i w_{ij}(\lambda_i \tau_{syn}) \ , \ net\_\sigma_j^2 = \sum_i (\frac{1}{2} w_{ij}^2)(\lambda_i \tau_{syn}) \ . \tag{4.12}$$

In accordance with $net_j = \sum_i w_{ij} x_i$, the input x_i of an artificial spiking neuron can be seen as~~$x_i = \lambda_i \tau_{syn}$, see Figure ??(a).~~ :

$$x_i = \lambda_i \tau_{syn} \ . \tag{4.13}$$

Figure 4.11 illustrates the process that an NSP-modelled artificial spiking neuron takes the input vector $\mathbf{x}$ which is converted from the input firing rate $\boldsymbol{\lambda}$; transforms the weighted sum net_x_j and $net\ \sigma_j^2$ to the abstract output y_j; and scales up y_i with the factor S to the output firing rate λ_j.

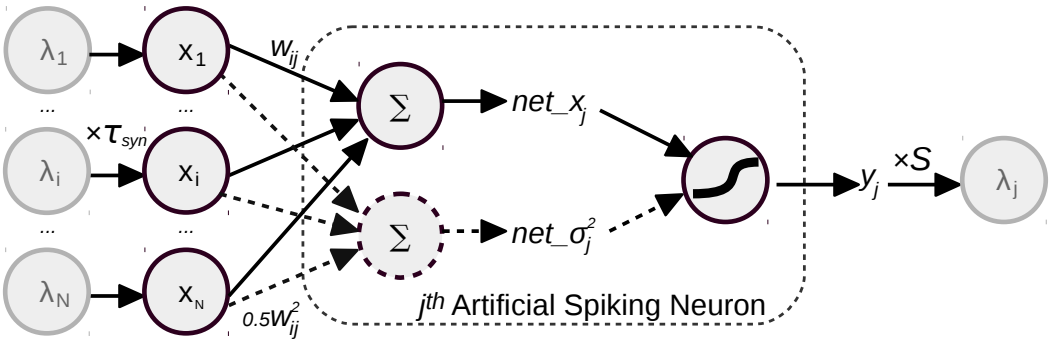


Figure 4.11: An artificial spiking neuron modelled by NSP. A spiking neuron takes firing rate λ_i as its input, which then forms the abstract numerical equivalence by multiplying the synaptic constant: $x_i = \lambda_i \tau_{syn}$. The NSP transforms the noisy current influx including both the mean (net_x_j, solid lines) and the variance $net\ \sigma_j^2$ (dashed lines) to the abstract firing rate y_j. Finally, the output of the NSP is mapped to the physical units, firing rates (λ_j) in Hz, by multiplying S.

If instead of multiplying every input firing rate λ_i by τ_{syn} (left of Fig. ~~??(a))~~ 4.11) we do it in every output firing rate after λ_j, (right of Fig. ~~??(b))~~ 4.12) we obtain the same neuron model and structure as a typical neuron in ANNs, see Figure ~~??(c)~~4.10, that neurons take ~~x~~ $\mathbf{x}$ as input and output abstract value y.

The only difference lies in the activation function where the artificial spiking neuron takes PAF, which is a simple linearly-scaled activation function with a parameter p~~, which~~ . The parameter is determined by the product of the scaling ~~parameter~~ factor S and the synaptic time constant τ_{syn}:

$$y = p \times f(x) = S \times \tau_{syn} \times f(x) \ , \tag{4.14}$$

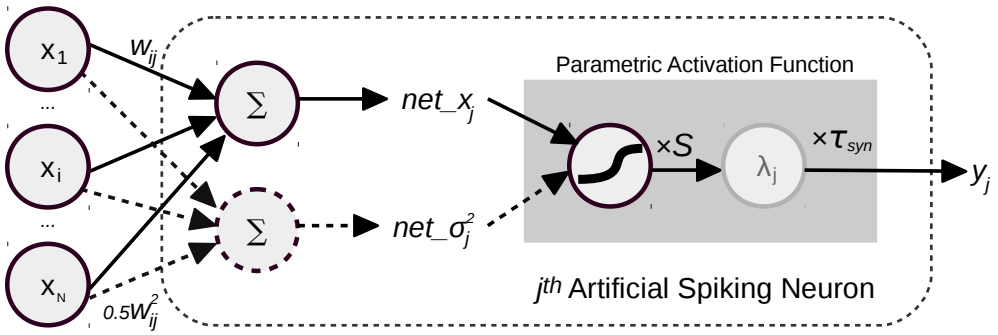


Figure 4.12: An artificial spiking neuron modelled by PAF-NSP, whose input and output are numerical values, equivalent to those of ANNs. PAF includes the scaling factors S and the synaptic time constant τ_{syn} in the combined activation function, which links the firing activity of a spiking neuron to the numerical value of ANNs.

~~and its derivative function~~ where $f(x)$ represents a typical conventional activation function, e.g. ReLU.

The derivative function of PAF, which is used for back propagation, is:

$$\frac{\partial y}{\partial x} = p \times f'(x) = S \times \tau_{syn} \times f'(x) \ . \tag{4.15}$$

~~Excitingly,~~ PAF not only allows NSP to model spiking LIF neurons on ANNs. Once the parameter p is acquired the PAF can be generalised to other Relu-like activation functions. Because Softplus and NSP will both converge to ReLU, when the input increases, so they can share the scaling factor p. Note that the calculation of noise level is not necessary for other activation functions, thus they only take the mean of the current influx as the input (the solid lines in Figure ~~??~~4.12). So the noise level can be set to a constant for Softplus or considered as 0 for ReLU.

It is also significant to transform numerical values of training and testing data to firing rates in the first/last layer of the SNN. In order to keep the firing rate in a valid range of an LIF neuron, e.g. less than the maximum firing rates of $\lambda_{max} = 1/\tau_{refrac}$, we can scale the labelling data of the last layer by multiplying λ_{max}/S during training. Thus according to PAF (Equation 4.14), the maximum firing rate of such an output neuron would be $1 * \lambda_{max}/S * S = \lambda_{max}$. We can certainly choose a much lower rate of λ_{max}, say 200 Hz, to keep the NSP fit to the actual LIF response activities better, since the parameters of PAF are curve-fitted to a limit working range of output firing rates. For the input layer, it is the easiest to keep the original abstract values as $\boldsymbol{x}$; then in the SNN test, we divide $\boldsymbol{x}$ by τ_{syn} to get the input firing rates of Poisson spike trains, see Equation 4.13. But, it is also flexible to linearly map the numerical values

to a range of firing rates by multiplying K Hz. Then, we use $\boldsymbol{x} \times K \times \tau_{syn}$ as the new input of the training network; and $\boldsymbol{x} \times K$ as firing rates of spike trains in SNN testing.

4.4.3 Training Method

The simple idea of PAF presented in the previous section allows the use of common ANN training methods to obtain SNN-compatible weights. Consequently, training SNNs can be done in three simple steps:

1. Calibrate the parameters ~~(k, S)~~ (k, b, S) for Noisy Softplus which models the response firing rates of LIF neurons, thus to estimate the parameter $p = S \times \tau_{syn}$ for PAFs ~~. Since (k, S)~~ and to set the LIF parameter $I_{offset} = b$. Since (k, b, S) are solely dependent on the biological configurations of an LIF neuron, the same p can be shared with different activation functions and repeatedly used for various network architectures and applications.

2. Train any feed-forward ANN with a PAF version of a ReLU-like activation function. Training compatibility allows us to choose computationally simple activation functions to increase training speed.

   As stated in Section 3.2, the Backpropagation algorithm updates weights using the optimisation method, stochastic gradient descent, to minimise error between the labels and the predictions from the network.

3. Transfer the trained weights directly to the SNN, which should use the same LIF characteristics used in Step 1.

4.4.4 Fine Tuning

As stated above, we can train the network with any PAF version of conventional ReLU-like activation functions, and then fine-tune it with PAF-NSP in the hope of improving the performance of the equivalent SNN by closely modelling the spiking neurons with NSP. Additionally, we add a small number, for example 0.01, to all the binary values of the labels on the training data. Although binary labels enlarge the disparities between the correct recognition label and the rest for better classification

capability, spiking neurons seldom stay silent even with negative current influx, thus setting labels to 0 is not practical for training SNNs. Therefore, adding an offset relaxes the strict objective function of predicting exact labels with binary values.

There are two aspects to the fine tuning which make the ANN closer to SNNs: Firstly, using the NSP activation functions causes every single neuron to run at a similar noise level as in SNNs, thus the weights trained by other activation functions will be tuned to fit closer to SNNs. Secondly, the output firing rate of any LIF neuron is greater than zero as long as noise exists in their synaptic input. Thus adding a small offset on the labels directs the model to approximate practical SNNs. The result of fine tuning on a ConvNet will be demonstrated in Section 4.5.3.

4.5 Results

Finally, the proposed generalised SNN training method is put into practice. We train a 6-layer ConvNet with PAF-NSP, and transfer the tuned weights into an equivalent SNN. The detailed description of the experiment can be found in Section 4.5.1. We then observe the single neuronal activities of the trained SNN (Section 4.5.2), compare the learning (Section 4.5.3) and recognition performance (Section 4.5.4) between activation functions, and estimate the power consumption of the SNN running on neuromorphic hardware (Section 4.5.5).

4.5.1 Experiment Description

A spiking ConvNet was trained on the MNIST [LeCun et al., 1998] dataset, using the generalised SNN training method stated above. The architecture (784-6c-6p-12c-12p-10fc illustrated in Section 3.2) contains 28×28 input units, followed by two convolution-pooling layers with 6 and 12 convolutional kernels each, and 10 output neurons fully connected to the last pooling layer to represent the classified digit.

To train the ConvNet, firstly, we estimated parameter p for PAFs given LIF configurations listed in ~~Talbe~~Table 4.1 and ~~$\tau_{syn} = 5$ ms, $p = S \times \tau_{syn} = 1.005$, where $(k = 0.30, S = 201)$~~ $\tau_{syn} = 0.005$ s, $p = S \times \tau_{syn} = 1.085$, where $(k = 0.31, b = 0.1, S = 217)$ were calibrated using NSP. Secondly, the training employed PAFs with three core activation functions: ReLU, Softplus and NSP to compare their learning and recognition

performance. The weights were updated using a decaying learning rate, 50 images per batch and 20 epochs. Finally, the trained weights were then directly transferred to the corresponding spiking ConvNets for recognition test on the SNN simulator, NEST [Gewaltig and Diesmann, 2007]. To validate the effect of fine tuning, we took another training epoch to train these models with PAF-NSP with data labels shifted by +0.01. Then the weights were also tested on SNN simulations to compare with the ones before fine-tuning.

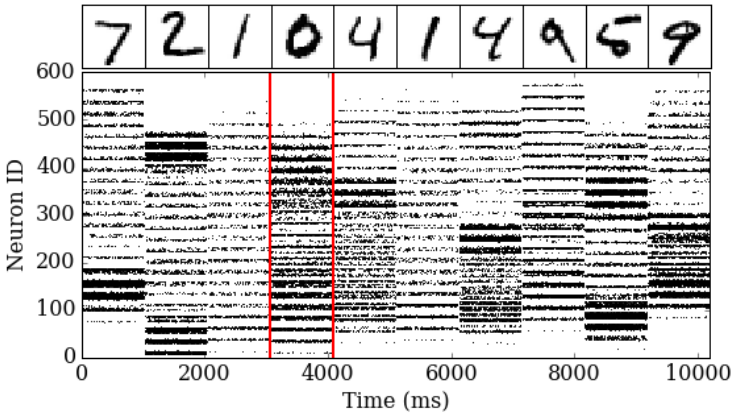
At the testing stage, the input images were converted to Poisson spike trains [Liu et al., 2016] and presented for 1 s each. The output neuron which fired the most indicated the classification of an input image.

4.5.2 Single Neuronal Activity

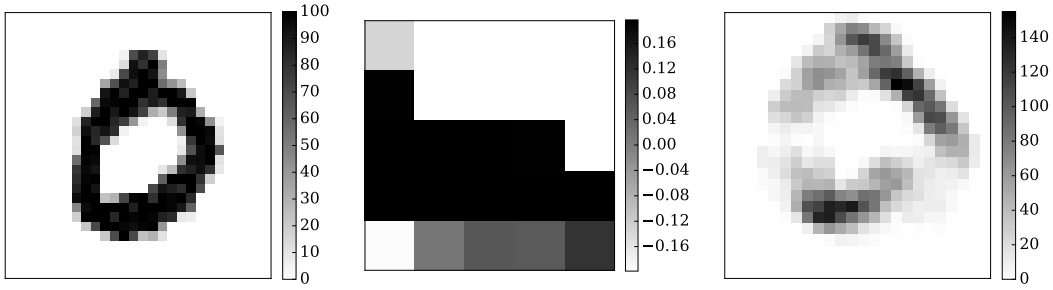
To validate how well the NSP activation fits the response firing rate of LIF neurons in SNNs, we simulated one of the PAF-NSP trained ConvNets on NEST. Ten testing images were presented with spike trains whose firing rates were calculated as: $\boldsymbol{\lambda} = \boldsymbol{x}/\tau_{syn}$. The inputs were convolved with a trained 5×5 kernel, and the output firing rates of the spiking neurons were recorded, see Figure 4.13.

The recorded firing rates are compared to the predictions of these PAFs: $\lambda' = S \times f(x) = y/\tau_{syn}$, see Figure 4.14. The estimated spike counts using NSP fitted the real recorded firing rate much more accurately than ReLU and Softplus. The Euclidean distances, $\sqrt{\sum_j (\lambda'_j - \lambda_j)^2}$, between the spike counts and the firing rates predicted by NSP, ReLU and Softplus were ~~184.57, 361.64 and 1102.76~~ 180.59, 349.64 and 1293.99 respectively. We manually selected a static noise level of 0.45 for Softplus, whose estimated firing rates located roughly on the top slope of the real response activity. This resulted in a longer Euclidean distance than using ReLU, since most of the input noisy currents were of relatively low noise level in this experiment. Hence, the firing rate driven by the lower noise level is closer to the ReLU curve than to Softplus.

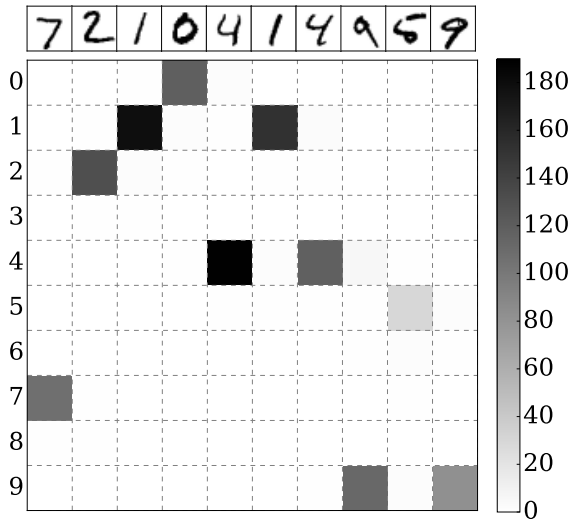
Note that there is a visible mismatch between the actual firing rates and the model estimation on the lower right region in Figures 4.14(a, c), where the blue dots (actual spike counts) fall lower the bound of ReLU. It is consistent with the statement in Section 4.3.3 that the LIF response activities does not fit into the NSP function when the noise level is low (approaching to 0). However, the minor mismatch does not result



(a) 10 input digits presented in Poisson spike trains.



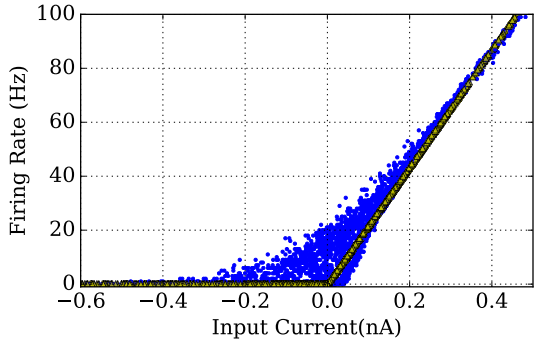
(b) Pixel firing rates in Hz. (c) ~~5x5~~ 5×5 weights kernel (d) Convolved output in Hz. (synaptic efficacy in nA).



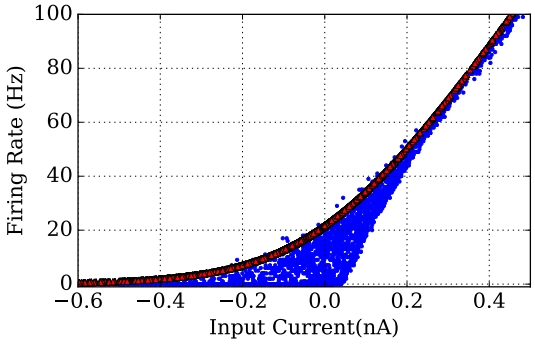
(e) Output firing rates of classification neurons.

Figure 4.13: Images presented in spike trains convolve with a weight kernel. (a) The 28×28 Poisson spike trains in raster plot, representing 10 digits in MNIST. (b) The firing rate of all the 784 neurons of the fourth image, digit '0', is plotted as a 2D image. (c) One out of six of the trained kernels (5×5 size) in the first convolutional layer. (d) The spike trains plotted as firing rate of the neurons in the convolved 2D map. (e) Output firing rates for recognising these digits.

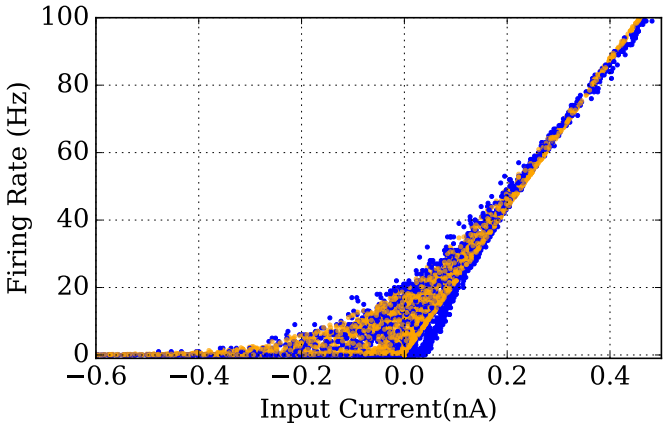
in poor performance on classification accuracy.



(a) Recorded data vs. ReLU.



(b) Recorded data vs. Softplus.



(c) Recorded data vs. NSP.

Figure 4.14: The recorded firing rate of the convolution of the same kernel with 10 images in SNN simulation, compared to the firing rate prediction by $S \times f(x)$. NSP (a) fits to the neural response firing rate of LIF neurons more closely than ReLU (b) and Softplus (c).

Figure 4.13(e) demonstrates the output firing rates of the 10 recognition neurons when tested with the digit sequence. The SNN successfully classified the digits where the correct label neuron fired the most. We trained the network with binary labels on the output layer, thus the expected firing rate of correct classification was ~~$1/\tau_{syn} = 200$~~$1 \times S = 217$ Hz according to Equation ~~??~~4.13. The firing rates of the recognition test fell to the valid range~~around 0 to 200 Hz~~. This shows another advantage of the NSP that we can estimate the firing rate of an SNN by $S \times f_{NSP}(x)$ running its equivalent ANN, instead of simulating SNNs. Moreover, we can constrain the expected firing rate of the top layer, thus preventing the SNN from exceeding its maximum firing rate, for example 1 KHz when the time resolution of the simulation is set to 1 ms.

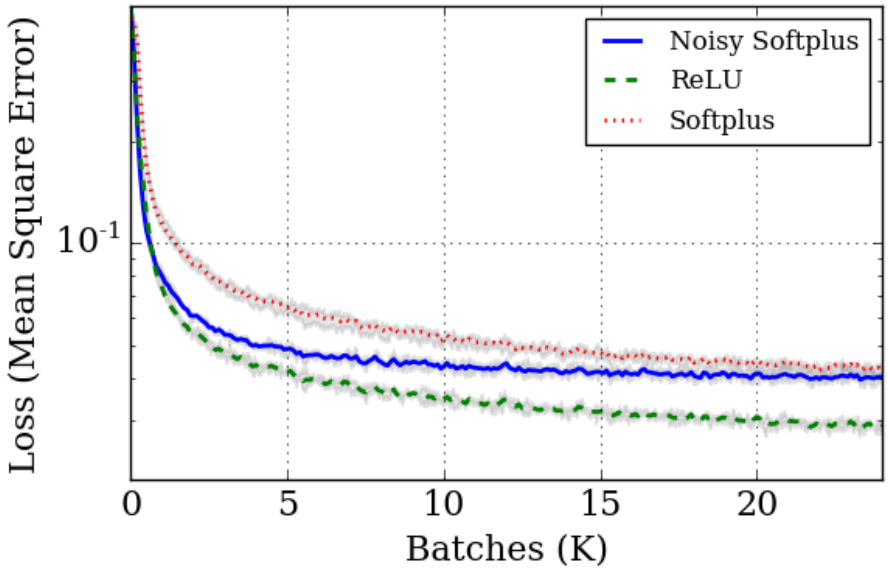


Figure 4.15: Comparisons of loss during training using Noisy Softplus, ReLU and Softplus activation functions. Bold lines show the average of three training trials, and the grey colour illustrates the range between the minimum and the maximum values of the trials.

4.5.3 Learning Performance

Before looking into the recognition results, it is significant to see the learning capability of the novel activation function, NSP. We compared the training using ReLU, Softplus, and NSP by their loss during training averaged over three trials, see Figure 4.15. ReLU learned fastest with the lowest loss, thanks to its steepest derivative. In comparison, Softplus accumulated spontaneous 'firing rates' layer-by-layer and its derivative may experience gradual or even vanishing gradients during back propagation, which result in more difficult training. The recognition performance of NSP lay between these two~~in terms of loss and learning speed~~. The loss stabilised to the same level as Softplus, because of the same problem of gradual gradients.

However, the learning stabilised the fastest using NSP which may be caused by the accurate modelling of the noise. Similar findings have shown that networks with added noise, e.g. dropout [Srivastava et al., 2014], are much faster to train. The result is promising that NSP may shorten the training time.

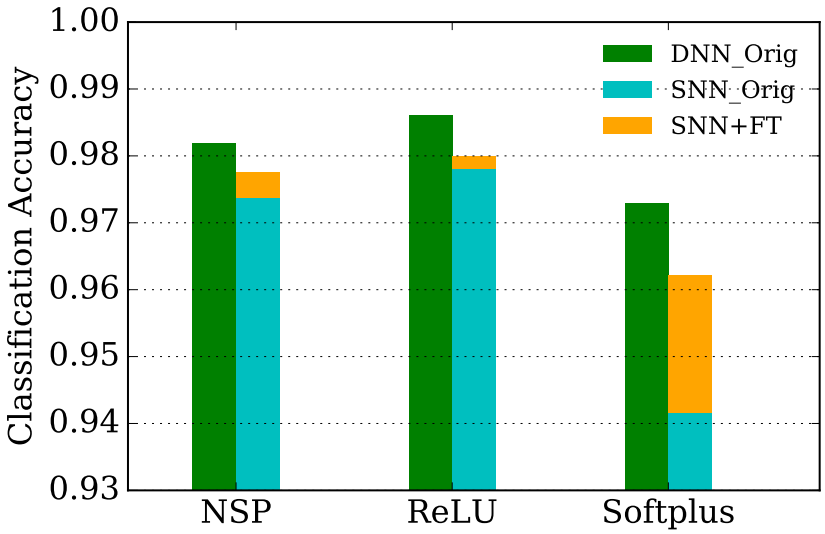


Figure 4.16: Classification accuracy. The trained weights were tested using the same activation function as training (DNN_Orig), then transferred to an SNN and tested on NEST simulation (SNN_Orig), and finally fine-tuned to be tested on SNN (SNN_FT) as well.

4.5.4 Recognition Performance

Classification Accuracy

The classification errors for the tests are investigated by comparing the average classification accuracy among three trials, shown in Figure 4.16. At first, all trained models were tested on the same artificial neurons as used for training in ANNs, and these experiments were called the 'DNN' test since the network had a deep structure (6 layers). Subsequently, the trained weights were directly applied to the SNN without any transformation, and these 'SNN' experiments tested their recognition performance on the NEST simulator. From DNN to SNN, the classification accuracy declines by 0.80%, 0.79% and 3.12% on average for NSP, ReLU and Softplus. The accuracy loss is caused by the mismatch between the activations and the practical response firing rates, see examples in Figure 4.14, and the strict binary labels for NSP and Softplus activations. Fortunately, the problem is alleviated by fine tuning which increases the classification accuracy by 0.38%, 0.19% and 2.06%, and results in the total loss of 0.43%, 0.61%, and 1.06% respectively. Softplus benefited the most from fine tuning, since the huge mismatch (Figure 4.14(c)) of response firing rate is greatly corrected. The improvement of NSP is obtained from the offset on the labels which helps the network to fit practical SNNs. As the recognition performance of ReLU is already

high, there is little room for improvement. Even though the fine-tuning procedure does its job, the gain in accuracy is the smallest for this activation function.

The most efficient training in terms of both classification accuracy and algorithm complexity, takes PAF-ReLU for ANN training and PAF-NSP for fine tuning. The best classification accuracy achieved by a larger spiking ConvNet (784-16c-16p-64c-64p-10fc) was 99.07% after fine tuning, a 0.14% drop from the ANN test (99.21%). The network reached the recognition rate of 98.7% even without fine tuning, thus we suggest to make fine tuning an optional step for training.

Comparisons in Literature

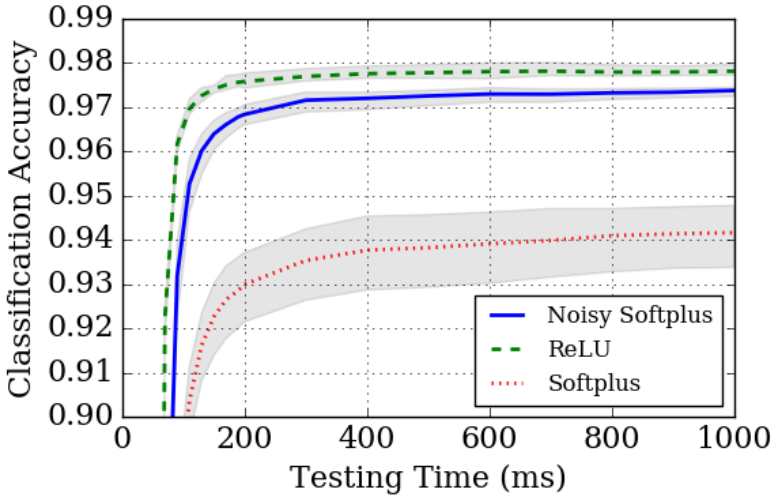
It is useful to compare with existing SNN training methods shown in Table 4.2 where we order them on the computation complexity (descending). The generalised training method presented here uses simple abstract activation functions, e.g. PAF-ReLU; it requires no modulations of trained weights to adapt to SNNs, but uses a single *optional* additional processing of fine tuning; The training method is well fitted to biologically-plausible LIF neurons, which are supported by most neuromorphic platforms. Regarding the classification accuracy, it achieves the state-of-the-art performance of SNNs and compares favourably with all the other methods using LIF neurons. The combination of these features compose a method with exceptional performance and ease of use for training SNNs.

Recognition Time

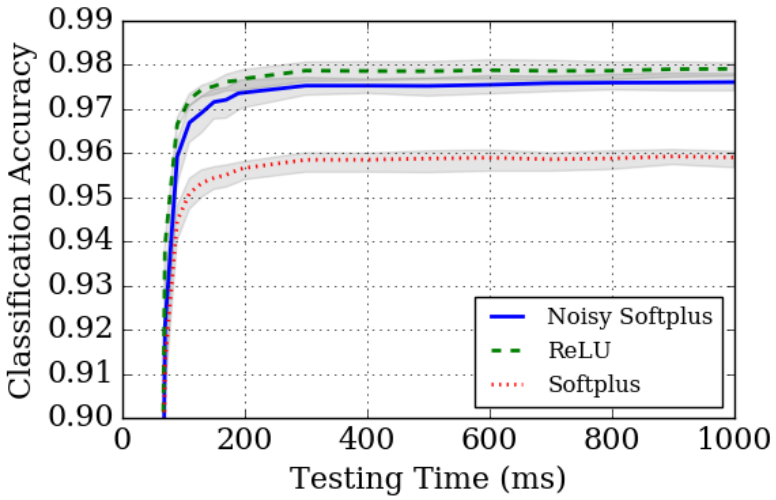
As it is a major concern in neuromorphic vision, the recognition performance over short response times is also estimated in Figure 4.17. After fine tuning, Softplus significantly reduced the mismatch since the randomness among the three trials shrinks to a range similar to other experiments. Fine tuning also improved its classification accuracy and the response latency. Notice that all of the networks trained by three different activation functions showed a very similar stabilisation curve, which means they all reached an accuracy close to their best after only 300 ms of biological time.

Table 4.2: SNN training methods comparison.

Method	Activation Function	Biologically-plausibility	Additional Processing	Weights Conversion	Classification Accuracy(%)
[Jug et al., 2012]	Siegert	**Yes**	**No**	**No**	94.94 [Stromatias et al., 2015a]
[Hunsberger and Eliasmith, 2015]	Soft LIF	**Yes**	Noisy inputs and activations	**No**	98.37
[Diehl et al., 2015b]	**ReLU**	No	Dropout	Yes	**99.1**
This Chapter	**PAF** (p×ReLU)	**Yes**	**No** or fine tune	**No**	98.72 **99.07 (fine tune)**



(a) Before fine tuning



(b) After fine tuning.

Figure 4.17: The classification accuracy of 3 trials (averaged in bold lines, grey shading shows the range between minimum to maximum) over short response times, with (a) trained weights before fine tuning, and (b) after fine tuning.

4.5.5 Power Consumption

Noisy Softplus can easily be used for energy cost estimation for SNNs. For a single neuron, the energy consumption of the synaptic events it triggers is:

$$\begin{aligned} E_j &= \lambda_j N_j T E_{syn} \\ &= \frac{y_j N_j T E_{syn}}{\tau_{syn}} \end{aligned} \quad , \tag{4.16}$$

where λ_j is the output firing rate, N_j is the number of post-synaptic neurons it connects to, T is the testing time, and E_{syn} is the energy cost for a synaptic event of some

specific neuromorphic hardware, for example, about 8 nJ on SpiNNaker [Stromatias et al., 2013]. Thus to estimate the whole network, we can sum up all the synaptic events of all the neurons:

$$\sum_j E_j = \frac{TE_{syn}}{\tau_{syn}} \sum_j y_j N_j. \tag{4.17}$$

Thus, it may cost SpiNNaker 0.064 W, 192 J running for 3,000 s with synaptic events of $8 \times 10^6/s$ to classify 10,000 images (300 ms each) with an accuracy of 98.02%. The best performance reported using the larger network may cost SpiNNaker 0.43 W operating synaptic event rate at $5.34 \times 10^7/s$, consume 4271.6 J to classify all the images for 1 s each.

4.6 Summary

We presented a generalised off-line SNN training method to tackle the research problem of equipping SNNs with equivalent cognitive capability to ANNs. This training procedure consists of three simple stages: first, estimate ~~parameter p~~ parameters for PAF using NSP; second, use a PAF version of conventional activation functions for ANN training; third, the trained weights can be directly transferred to the SNN without any further transformation. ~~Another important feature of accurately modelling LIF neurons in ANNs is the acquisition of spiking neuron firing rates. These will aid deployment of SNNs in neuromorphic hardware by providing power and communication estimates, which would allow better usage or customisation of the hardware platforms.~~

Regarding the generalisation, the training not only uses popular activation functions in ANNs, e.g. ReLU, but also targets standard LIF neurons which are widely used on neuromorphic hardware. Therefore, the proposed method remarkably simplifies the training of AI applications for neuromorphic hardware; thereby paving the way to energy-efficient AI on the brain-like computers: from neuromorphic robots to clusters. Moreover, it lowers the barrier for AI engineers to easily access neuromorphic hardware without the need to understand SNNs or the hardware. Furthermore, this method requires the least computation complexity while performing most effectively among existing algorithms. In terms of classification/recognition accuracy, the performance of ANN-trained SNNs is nearly equivalent to ANNs, and the performance loss can be partially solved by fine tuning. The best classification accuracy of 99.07%

using LIF neurons in a PyNN simulation outperforms state-of-the-art SNN models of LIF neurons and is equivalent to the best result achieved using IF neurons. Another important feature of accurately modelling LIF neurons in ANNs is the acquisition of spiking neuron firing rates. These will aid deployment of SNNs in neuromorphic hardware by providing power and communication estimates, which would allow better usage or customisation of the hardware platforms.

Chapter 5

On-line SNN Training with Spike-Based Rate Multiplication

In the previous chapter we argued that deep Spiking Neural Networks (SNNs) can be trained simply off-line on equivalent Artificial Neural Networks (ANNs) and are equally capable of classifying hand written digits as are deep ANNs. This chapter continues the discussion of the main research problem to narrow the gap of cognitive capabilities between spiking and conventional neural networks. Instead of transforming off-line trained ANN models into SNNs, we explore on-line approaches which directly modulate the plastic synapses between spiking neurons in a biologically-plausible manner.

5.1 Introduction

Before we investigate the proposed method, it is helpful to clarify what on-line learning addresses in the context of this thesis and what special features exist in SNN training. Researchers in neuromorphic engineering seem to take the term 'on-line training' for granted [Neil, 2013; Neftci et al., 2013] without describing it clearly. The on-line approach exploits biologically-plausible learning rules, e.g. Spike-Timing-Dependent Plasticity (STDP). Therefore, the modulation of the synaptic weights is event-driven by the spikes and operates in biological real time. On the contrary, 'off-line' training usually takes place on an equivalent ANN and the network parameters are tuned using traditional algorithm, e.g. gradient descent.

Therefore, some special features appear in the on-line systems. Firstly, on-line systems 'learn through play' in that there is no separation between a training and a testing phase. Typically, the systems learn and improve their capability continuously as more data is fed into them. Secondly, they 'live and learn' which means on-line learning never stops. Thus, it is easy for an on-line system to learn a new task, by simply providing different data. However, once a model is trained off-line, it remains fixed and stops learning. The brain is a natural on-line system, thus bringing its learning techniques to SNNs will equip neuromorphic computers with genuine learning capability, moving towards Neuromorphic Cognition. ~~Hence, thirdly,~~

One of the practical problems, the stability-plasticity dilemma [Grossberg, 1987], is a typical example which only exists in artificial systems but not in the brain. Off-line trained systems cannot learn anything new, whereas on-line ~~approach exploits biologically-plausible learning rules, e. g. Spike-Timing-Dependent Plasticity (STDP). Therefore, the modulation of the synaptic efficacy is event-driven by the spikes and operates in biological real time.~~ learning systems easily lose their previous knowledge. However, the brain intuitively achieves both stability and plasticity simultaneously; it maintains gained knowledge while being plastic in respond to new input. Hence, there will be important lessons, such as controlling learning, protecting memories/memory segmentation, and etc., to learn from the brain before an on-line SNN system delivers genuine learning capability.

The main difficulty in proposing effective on-line methods is the lack of knowledge about learning in the brain. The STDP learning rule, presented by Bi and Poo [1998] (see Section 2.2.4 for detail), and its variations make up the majority of the biologically plausible on-line learning methods. However, the most common training algorithm for an ANN, Backpropagation (BP), is difficult to transform into STDP, since STDP usually works locally with a teaching signal in supervised learning; but, BP does not provide the teaching targets for all the hidden units of a network. Therefore, existing on-line approaches, including our proposed method, favour greedy layer-wise unsupervised training in Deep Learning [Hinton et al., 2006].

Another problem is to transform numerical calculations of weight changes into spike-based synaptic learning rules. Existing methods suffer from performance loss due to imprecise translation. To address this problem, we propose the Spike-based

Rate Multiplication (SRM) method. This algorithm transforms numerical calculations of weight tuning accurately into precise parameter configurations of equivalent STDP rules. We select multiplication to transform into SNNs for the reason that it is the core operation in the greedy layer-wise training of Autoencoders (AEs) and Restricted Boltzmann Machines (RBMs).

Moreover, correlations between spike trains bring down the learning performance drastically after it peaks, thus the learning has to be manually stopped to avoid performance drop. The problem makes 'live and learn' far from achievable in practice. Therefore, we put forward four methods to reduce the spike correlations while learning, thereby providing potential solutions to achieving genuine on-line learning on neuromorphic hardware.

To begin with in Section 5.2 we explore the research question of on-line, event-based deep SNN training in the literature. In Section 5.3 we describe SRM mathematically, and explain the factors influencing the accuracy of the method. We then argue why the learning algorithm is suited to train spiking Autoencoders (SAEs) and Spiking Restricted Boltzmann Machines (SRBMs) in Section 5.4. During the research we encountered the problem of correlated spikes, and propose solutions to decorrelate spike trains in Section 5.5. Finally, experiments in Section 5.6 show that this on-line training method achieves better performance than existing algorithms and approaches the same, sometimes superior performance of the equivalent non-spiking methods using detailed comparisons on the MNIST dataset.

5.2 Related Work

The first on-line training algorithm using unsupervised Deep Learning was event-based Contrastive Divergence (evtCD) [Neil, 2013] for SRBMs, which established the feasibility of applying the STDP rule to an approximate CD algorithm. A review of training methods for RBM algorithms is given in Section 3.4. The evtCD method divided the learning process into potentiation and depression parts, which forms the basic design of related research. Although it was also the first to intuitively transform numerical calculations to STDP rules, as an initial attempt mathematical estimation of the parameters was mainly ignored. Therefore the best classification performance achieved

was only about 81.5% on the MNIST task, using purely spiking neurons. Neftci et al. [2013] derived the division of potentiation and depression to their event-based CD algorithm, but conducted both parts on the same neural populations which was more biologically plausible. Therefore, a global signal was needed to differentiate the potentiation and depression process. The rhythmic oscillation of these processes generated a particular form of neural sampling [Petrovici et al., 2013]. The work focused on the statistical aspect of the event-driven sampling, and was extended to the later work on Synaptic Sampling Machines (S2Ms) [Neftci et al., 2016] using stochastic synapses. The classification accuracy on MNIST was 91.9%, 1.7% lower than the conventional RBM, however was improved to 95.4% by the S2M using the edge tool of Deep Learning: dropout [Srivastava et al., 2014]. Another similar work on SRBM training [Burbank, 2015] applied the STDP rule to train SAEs in a very much biologically-plausible manner. This work aimed at constructing artificial machine learning algorithms close to biology, whereas the other works above focused on training spiking Deep Learning modules with recognition capability.

The work we propose was mostly inspired by the supervised time-based spiking learning rule, ReSuMe [Ponulak and Kasinski, 2010], where the output spikes of a post-synaptic neuron were trained to fire at the same times as the teaching spikes, see Figure 5.1. The teaching signal always potentiates the synaptic strength while the output spikes of the post-synaptic neuron always depress the connection, and the weight change Δw updates in accordance with the exponential STDP curve against the time difference between a post- and a pre-synaptic spike. Therefore, as illustrated in Figure 5.1, at the beginning the teaching spike fires earlier than the post-synaptic spike within an STDP window, τ_{win} which makes the weight potentiate more than it depresses; thus (shown in the middle of Figure 5.1) driven by a stronger synaptic weight, the post-synaptic neuron fires earlier than it is supposed to, making the weight depress a bit more than its potentiation; finally, the weight stays unchanged because the post-synaptic neuron fires coincidently with the teaching spike, thus the weight's potentiation cancels out the depression.

The inspiration of the proposed method is to provide a learning rule equivalent to ReSuMe on rate-encoded SNNs. The objective is to cause the post-synaptic neuron to fire at the frequency of the teaching spike train. Accordingly, the weight decreases

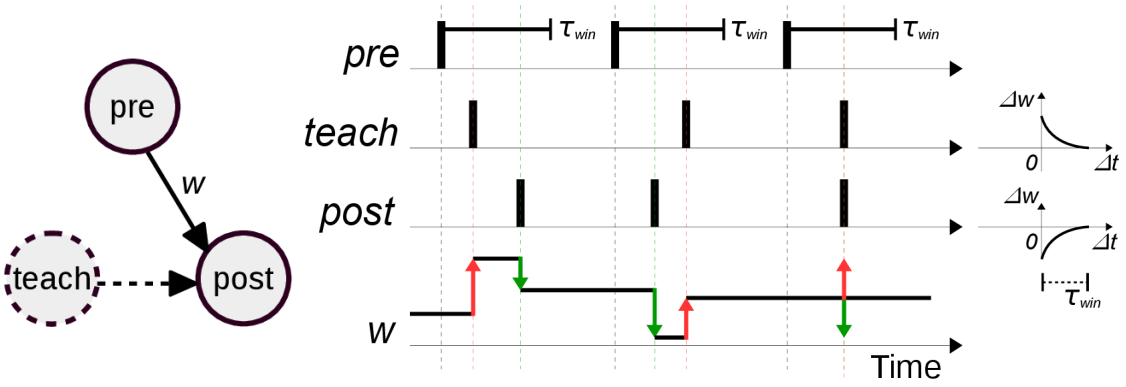


Figure 5.1: A pair of pre- and post-synaptic neurons trained by ReSuMe [Ponulak and Kasinski, 2010] with a teaching signal.

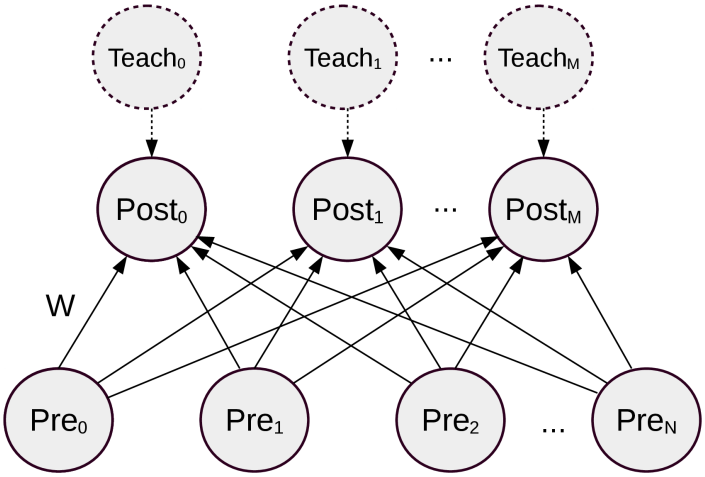


Figure 5.2: The architecture of an ADALINE network.

if the output neuron fires stronger than the teaching neuron, and vice versa. The synaptic strength stops changing when the output neuron fires at the same frequency as the teaching signal. In the following section, we will firstly describe this idea mathematically using the proposed term SRM, and equip the method with the ability to learn using a biological-plausible STDP rule.

5.3 Spike-based Rate Multiplication (SRM)

Following the idea of reconstructing teaching signals in rate-based networks, we firstly look into existing models in ANNs. If we see *pre*, *post* and *teach* individually as vectors, and w as a weight matrix of the all-to-all connections between *pre* and *post* in Figure 5.1, it will form the ADALINE (Adaptive Linear Element) [Widrow and Hoff, 1960] network, see Figure 5.2. The ~~'Post'~~ *'post'* neurons perform a weighted sum on

the input data, and the error between their output and the teaching data is propagated to update the weights, thus to train the network to generate the same output as the teacher. The learning algorithm was named Widrow-Hoff after the researchers:

$$\Delta w = \eta(teach - post)pre \ . \tag{5.1}$$

The right hand side of the equation can be seen as a subtraction of two multiplication operations, $teach \times pre$ and $post \times pre$, times a learning rate, η. Similarly, the unsupervised learning of AEs and RBMs has the same form of weight modulation, see Equations 3.11 and 3.22:

$$\Delta w = \eta(ab - cd) \ \ . \tag{5.2}$$

Especially for the training of AEs, the weight updates are the same as Equation 5.1 if using the Rectified Linear Unit (ReLU) as the activation function.

Therefore, multiplications are the core operations in the training of these rate-based ANN models. Thus we propose the SRM accurately to transform the product of rates to weight tuning of event-based, biologically-plausible learning in SNNs. There are a few steps to be followed: (1) represent the rate multiplication with simultaneous spikes generated from a pair of connected spiking neurons; (2) capture the simultaneous events in the weight change of the synaptic connection using the STDP learning rule. (3) precisely transform the learning rate η to parameters used in SRM.

1. Presenting rates with spikes

Firstly, the multiplier a and the multiplicand b are encoded into Poisson spike trains $s_a(t) = \{s_a(1), s_a(2), ..., s_a(T)\}$ and $s_b(t) = \{s_b(1), s_b(2), ..., s_b(T)\}$ with 1 ms resolution, where $s(t) = 1$ indicates a spike in the tth ms and $s(t) = 0$ means no spike. Secondly, a Poisson generator fires a sequence of spikes according to its firing rate, λ Hz, which is assigned linearly to the original multiplier/multiplicand with a scaling factor of K. Hence, the firing rate (λ_x) of the Poisson generator is K times the numerical value x, and can be approximated by the average spike count, $N_T(s_x)$, of the generated spike train $s_x(t)$ over time T ms:

$$\lambda_x = Kx \approx N_T(s_x) = \frac{1000}{T} \sum_{t=1}^{T} s_x(t) \ , \tag{5.3}$$

1000 is the scale factor to transform frequency per millisecond to frequency per second. Significantly, the approximation is more accurate as the observation time (T) grows

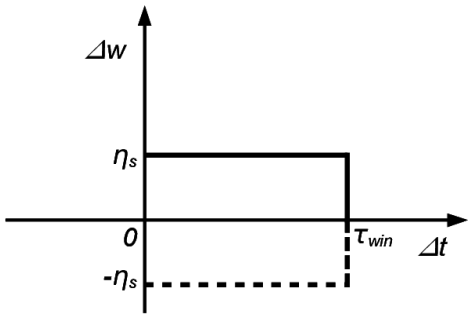


Figure 5.3: Rectangular STDP curve. If the time difference between the post-synaptic spike and the pre-synaptic spike lies in the window τ_{win}, then the synaptic weight will increase or decrease by η_s.

since more spikes are generated over time and the average spike count becomes more reliable. Thirdly, assuming $s_a(t)$ and $s_b(t)$ are independent Poisson spike trains the core definition of the rate multiplication of the pair of spike trains is as follows:

$$\lambda_a \lambda_b \approx N_{T_1}(s_a) N_{T_2}(s_b) = 10^6 \frac{\sum_{t_a=1}^{T_1} s_a(t_a)}{T_1} \frac{\sum_{t_b=1}^{T_2} s_b(t_b)}{T_2} \ . \tag{5.4}$$

Finally, if we constrain the length of the observing time T_2 to a short time window τ_{win} after each time step in T_1, the rate multiplication can be approximated with coincident spikes:

$$\lambda_a \lambda_b \approx \frac{10^6}{\tau_{dur} \tau_{win}} \sum_{t=1}^{\tau_{dur}} [s_a(t) \sum_{t_b=t}^{t+\tau_{win}} s_b(t_b)] \ , \tag{5.5}$$

where τ_{dur} replaces T to represent the length of generated spike trains during SNN simulation. Consequently, the rate multiplication can be conducted within only the local events in terms of time, although the accuracy may drop. Here, we define the SRM function:

$$\lambda_a \lambda_b \approx SRM(s_a, s_b) = \frac{10^6}{\tau_{dur} \tau_{win}} \sum_{t=1}^{\tau_{dur}} [s_a(t) \sum_{t_b=t}^{t+\tau_{win}} s_b(t_b)] \ . \tag{5.6}$$

2. Capturing coincident spikes with STDP.

The weight rise/drop according to a rectangular STDP curve can detect the spike events of a pair of neurons when a post-synaptic spike occurs coincidently (within τ_{win}), see Figure 5.3:

$$\begin{aligned} \Delta w &= STDP(s_a, s_b) \\ &= \eta_s \sum_{t=1}^{\tau_{dur}} [s_a(t) \sum_{t_b=t}^{t+\tau_{win}} s_b(t_b)] \ , \end{aligned} \tag{5.7}$$

where η_s represents the learning rate of the STDP rule. Therefore, the overall weight change during time τ_{dur} is determined by the number of coincident spikes of the pair of neurons, indicating the rate multiplication, and is described by SRM (Equation 5.6):

$$\begin{aligned} SRM(s_a, s_b) &= \frac{10^6}{\tau_{dur}\tau_{win}} \sum_{t=1}^{\tau_{dur}} [s_a(t) \sum_{t_b=t}^{t+\tau_{win}} s_b(t_b)] \\ &= \frac{10^6}{\tau_{dur}\tau_{win}\eta_s} STDP(s_a, s_b) \\ &= \frac{10^6}{\tau_{dur}\tau_{win}\eta_s} \Delta w \quad . \end{aligned} \tag{5.8}$$

3. Translating abstract numerical multiplication to SRM

If we separate the weight updates of Equation 5.2 to a positive Δw_+ and a negative part Δw_-, the weight tuning can be described as:

$$\begin{cases} \Delta w_+ = \eta ab \\ \Delta w_- = -\eta cd \end{cases} . \tag{5.9}$$

Thus, it is straight forward to estimate the parameter η_s to precisely transform the weight update from numerical calculations to spike-based learning rules:

$$\begin{aligned} \Delta w_+ = \eta ab &= \frac{\eta \lambda_a \lambda_b}{K^2} \\ &\approx \frac{\eta SRM(s_a, s_b)}{K^2} \\ &= \frac{\eta 10^6}{\tau_{dur}\tau_{win}\eta_s K^2} \Delta w_+ \ , \\ \text{to make the above equation work,} \quad 1 &= \frac{\eta 10^6}{\tau_{dur}\tau_{win}\eta_s K^2}, \\ \text{thereby,} \quad \eta_{s+} &= \frac{\eta 10^6}{K^2 \tau_{dur}\tau_{win}} \ , \\ \text{and similarly,} \quad \eta_{s-} &= -\frac{\eta 10^6}{K^2 \tau_{dur}\tau_{win}} \ . \end{aligned} \tag{5.10}$$

So far we have accurately transformed numerical calculations of weight tuning to precise parameters of the SRM, thereby to the parameters of the STDP rules. In another word, the final on-line learning implementation is to accurately set the learning rates η_{s+} and η_{s-} and the time window τ_{dur} for the STDP learning rule, and the weights of two connected spiking neuron will be continually modified given synchronous spikes of the pair of neurons.

Last but not least, we state the property of the SRM algorithm:

- the accuracy of SRM is mainly controlled by τ_{dur} and τ_{win}, where longer spike trains and a longer STDP window express the rate more reliably.
- in our spiking neural network both the multiplier and the multiplicand are presented only as rates, which are positive quantities. Thus a negative product is applied with weight decrease, η_{s-}.
- the multiplier and the multiplicand are interchangeable due to the independence of the spike trains.
- the accuracy is independent of the neural and synaptic models of the spiking neuron because the calculation relies only on the firing rate.

5.4 Training Deep SNNs

We have theoretically deduced the ~~equivalence~~ approximation of the numerical weight update to the one-line spike-based STDP rules. This section attempts to verify the SRM method in practice, thus we compare the learning performance of the conventional Deep Learning models to their spiking versions. We record the reconstruction performance of the models, and carefully observe the dynamics of weights modifications, the activities of the hidden and output neurons and the reconstruction loss. The experimental setup is described in Section 5.4.1, and the same experiments are carried out on all the training models for objective comparisons: AEs in Section 5.4.2, RBMs in Section 5.4.3, SAEs in Section 5.4.4 and SRBMs in Section 5.4.5.

5.4.1 Experimental Setup

There were two set-ups of the same network architecture (Figure 5.4) where ten visible neurons connected to a layer of ten hidden neurons symmetrically with all-to-all weights: in Experiment 1 (Exp1), ~~the values of all~~ each dimension (out of 10) of the input data ~~were~~ was 1, $input_1 = [1, 1, 1, ..., 1]$; and in Experiment 2 (Exp2), the ~~input data~~ values ranged from 0.1 to 1 linearly with steps of 0.1, $input_2 = [0.1, 0.2, 0.3, ..., 1]$. For both experiments, the same input data, either $input_1$ or $input_2$, is presented 5,000

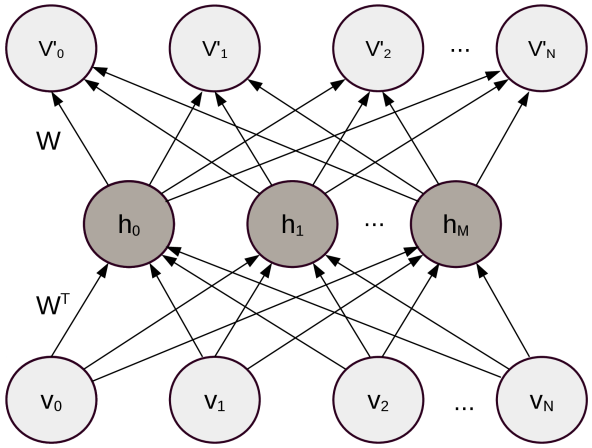
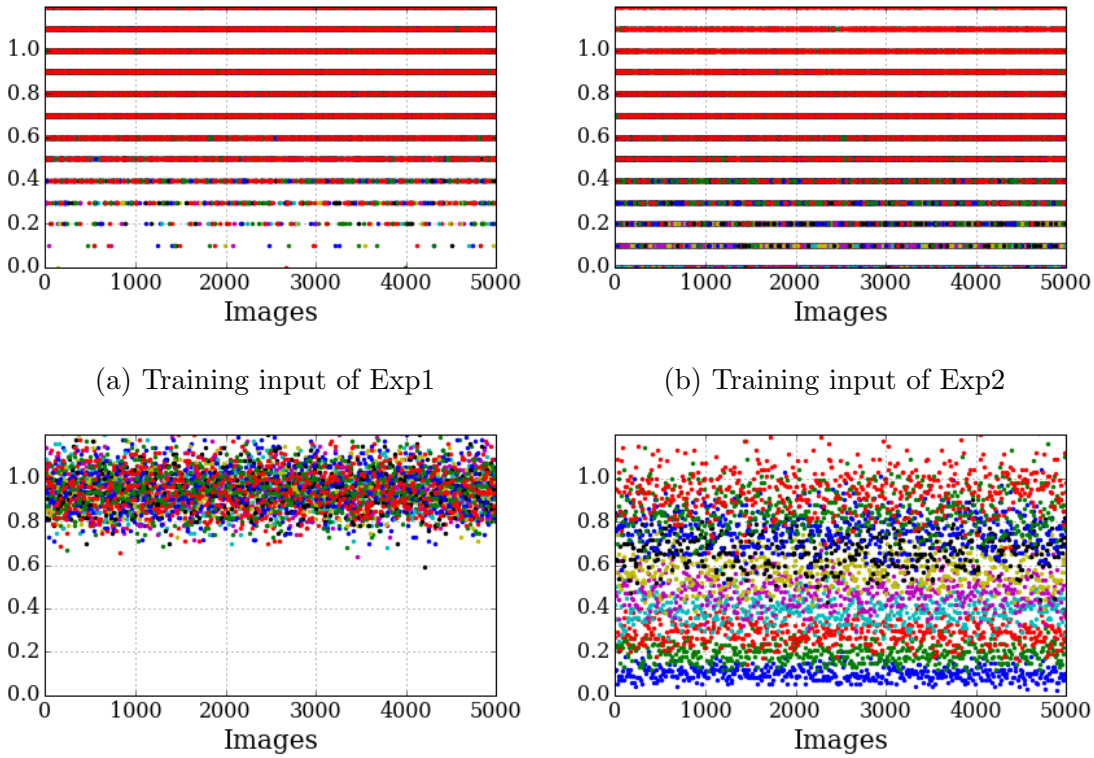


Figure 5.4: Symmetric weights connected between visible ($\mathbf{v}$) and hidden ($\mathbf{h}$) units in AEs and RBMs to reconstruct visible inputs, $\mathbf{v}'$.

times for every trial of training. These two experiments provided a close observation of the dynamic weight change given constant inputs. Exp1 showed how the network responded to the same input value and reconstructed it, while Exp2 demonstrated the influence of ranging input values and, more importantly, how different firing rates may affect the corresponding SNN. These experiments were run as baselines to compare with spike-based training on the features of weight convergence, reconstruction error and neural activities.

The initial weights were randomly generated with unified distribution from 0 to 0.01 and the learning rate of conventional models η, was set to 0.001, and for spike-based training was set to 0.0001. We kept the same initial weights for all the experiments thus providing an accurate comparison of the weight updates. The input ~~vector~~data vector of ten dimensions, seen as an image, repeatedly fed into the network $5,000$ times during training. Representing a data vector with the term 'image' helps us to better demonstrate the reconstruction task, and to have a unified expression as images in the MNIST dataset for later use. As a 'live and learn' system, there was no end to the learning, however for the purpose of observing the reconstruction performance, the weights were frozen every 10 steps of training and were validated on the testing data.

Initially, AEs and RBMs were trained with clean numerical values, then with noisy values generated by counting the spikes in Poisson spike trains. The noisy data was gathered from the SNN experiments in Section 5.4.4. All the SNN experiments used



(a) Training input of Exp1

(b) Training input of Exp2

(c) Testing input of Exp1

(d) Testing input of Exp2

Figure 5.5: Noisy input gathered from Poisson spike trains.

the same training and testing Poisson spike trains for the purpose of the unified experimental environment.

The ~~firing rates of the~~ input values were scaled up by the factor of $K = 100$~~, thus~~ Hz, thus the firing rates of the poisson spike trains are $\lambda_1 = [100, 100, 100, ..., 100]$ Hz and $\lambda_2 = [10, 20, 30, ..., 100]$ Hz. The spike count $N_{\tau_{dur}}$ of the generated Poisson spike train then transformed to the noisy input (NI) for use of conventional models: ~~$N_{\tau_{dur}} * 1000/(\tau_{dur} * K)$~~$N_{\tau_{dur}} times 1000/(\tau_{dur} \times K)$. The scaling factor $1,000$ converts ms to s, and the length of spike trains τ_{dur} was 100 ms when training, and $1,000$ ms for testing. The longer the spike trains, the less noisy the spike counts become. The NI can be seen as distorted data by adding Gaussian noise to the original value. Figure 5.5 shows the NI of Exp1 and Exp2; Every subfigure demonstrates $5,000$ data vectors, each data vector is shown as a column of coloured points, and each dimension of the input data is drawn with the same colour; every single data vector represents a trial of poisson spike trains, whose firing rates can be calculated by the values of the vector times $K = 100$ Hz; and the actual firing rates are recorded by randomly generated

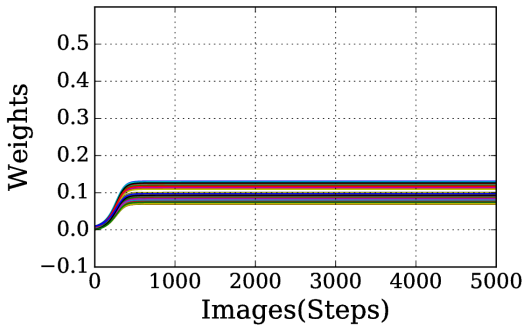
poisson spike trains given the expected firing rates of $\lambda_1 = [100, 100, 100, ..., 100]$ Hz and $\lambda_2 = [10, 20, 30, ..., 100]$ Hz; Since, the training poisson spike trains are 100 ms long, but the testing spike trains are 10 times longer, the actual firing rates are much more noisy in training data then in testing data. Hence, as shown in Figure 5.5, ~~the noisy training input~~ NI added a Gaussian noise with -0.05 mean and 0.29 variance for Exp1 to the clean data, and -0.02 mean and 0.22 variance for Exp2; while in terms of testing data, the means of the noise were the same, but the variances were much smaller: 0.09 for Exp1 and 0.07 for Exp2.

To compare all the following experiments, we present the results in the same template of figures: among such a set of figures, (a) and (b) depict the weight changes of the two experiment set-ups (Exp1 and Exp2) which are the most important output of the training method; (c) and (d) display the output of the visible units, the reconstruction of the input vector during testing; (e) and (f) draw the output of the hidden units during testing and assist the observation of weight change and the reconstruction; (g) and (h) intuitively show the loss (the mean squared error) and validate the accuracy of the reconstructions.

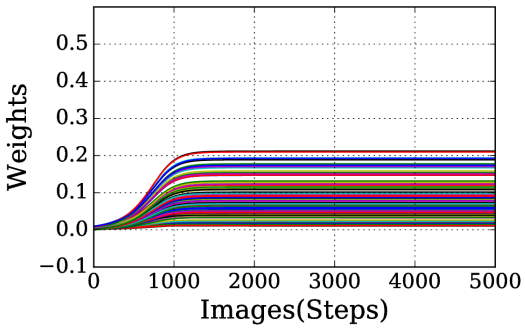
Following the experiments on conventional models, we propose the training methods for SAEs and SRBMs in detail. The same experiments were then applied to the on-line and spike-based SNN training.

5.4.2 Training AEs

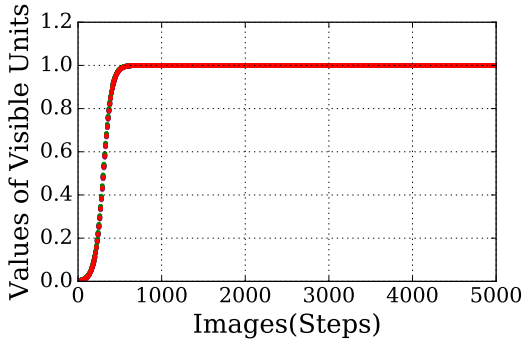
Using ReLU and Equation 3.11, we can easily train a layer of AEs with such a small network. For Exp1, Figure 5.6 shows the dynamics of the AE as more repeating images are presented through training. The reconstruction loss reduces exponentially to the limit of the computer's floating-point precision (Figure 5.6(g)), and reaches 10^{-4} using about 600 steps. From that point the weights, visible reconstruction, and the output of the hidden units nearly stabilise, see Figure 5.6(a,c,e). With the different input values of Exp2, the training runs slower, taking about $1,400$ steps to reach the same performance of 10^{-4} loss (Figure 5.6(h)). The reason for the slower training is due to the weaker input which also results in lower output of the hidden units comparing to Exp1, see Figure 5.6(f), thereby slowing down the weight increase. The reconstructions, shown in Figure 5.6(d), of smaller values stabilises earlier than those



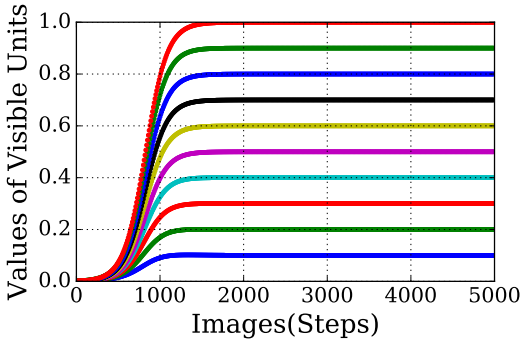
(a) Weights of Exp1



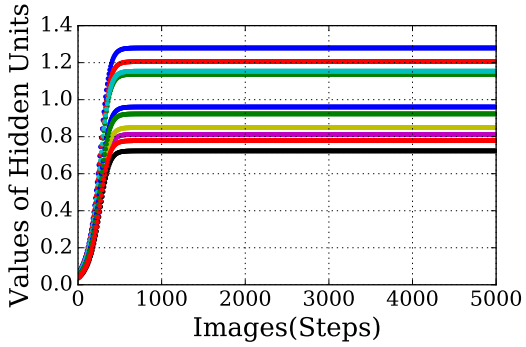
(b) Weights of Exp2



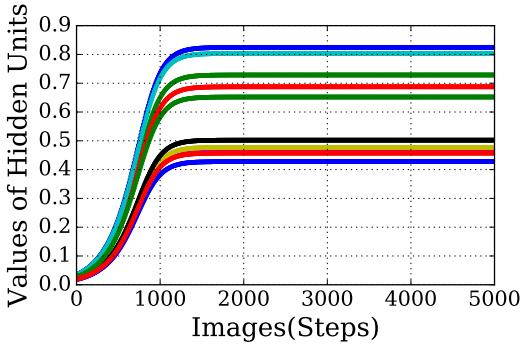
(c) Reconstruction of visible units in Exp1



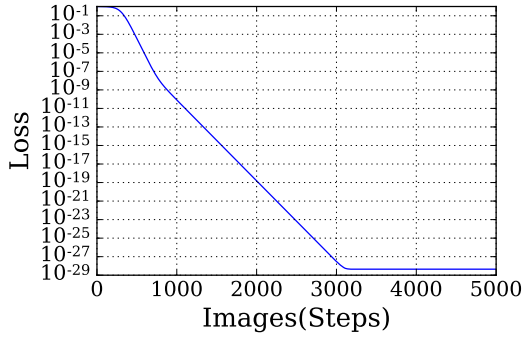
(d) Reconstruction of visible units in Exp2



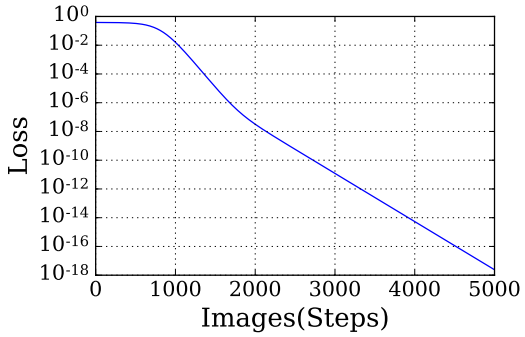
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2



(g) Output of hidden units in Exp1



(h) Output of hidden units in Exp2

Figure 5.6: Changes of weights, output of visible and hidden units, and mean squared error (loss) during the AE training of the reconstruction tests. Experiments 1) 10 visible units fully connected to 10 hidden units with input data of all 1s; 2) the same network fed with 10 values distributed linearly from 0.1 to 1.

of higher values, since the higher output of the reconstruction requires stronger weights and more accumulated weight updates.

The same experiments were repeated with noisy training and test data to provide a fair comparison with the spike-based Deep Learning modules. Figure 5.7 shows the effect of the NI, although the training stabilises at roughly the same time point. Firstly, it generates slight fluctuations in the weight change. Secondly, the reconstruction and the output of the hidden units are noisy compared to those of clean data. However, the noise is much reduced from the NI data compared to Figures 5.5 (c) and (d), indicating the robustness of AEs. Finally, the losses keep at a certain level and stops dropping. The loss in Exp2 is lower than Exp1, in other words the reconstruction on Exp2 is closer to the input data. This is mainly caused by the weaker noise level on the smaller input values which makes the data of Exp2 less noisy on average. So is the reconstruction, which leads to the lower level of loss.

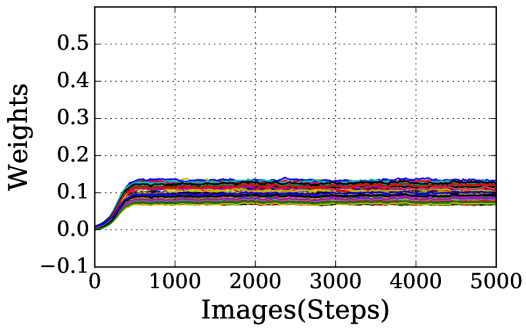
5.4.3 Training Noisy RBMs

Instead of using binary units and sigmoid activations for the RBM, we employed noisy ReLU (NReLU) units to construct an noisy Restricted Boltzmann Machine (nRBM) which was closer to biology and performed better in classification tasks [Nair and Hinton, 2010]. Leaving the learning algorithm unchanged, see Equation 3.22 in Section 3.4, the weight update is as follows:

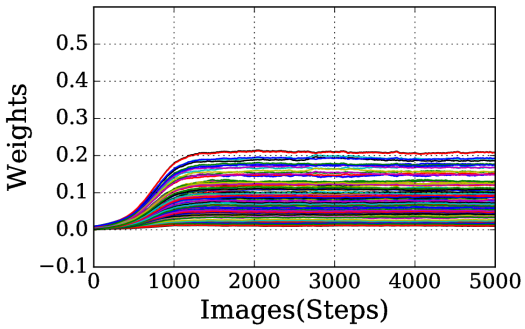
$$\Delta w_{ij} = \eta(h_i v_j - h'_i v'_j), \tag{5.11}$$

where a Gibbs sample comprises a pare of h'_i and v'_j, and the new sampling method NReLU is equivalent to generating multiple samples in the meantime and averaging them: $max(0, x + \mathcal{N}(0, \sigma(x))$. The lower the variance of the normal distribution, the more samples are taken for averaging; zero variance (equivalent to ReLU) is used when unlimited samples are generated, but at the same time the sampling itself loses the randomness. In our experiments the variance of the normal distribution used in NReLU is 0.1 during training.

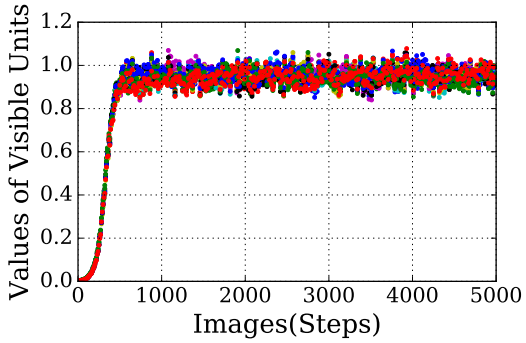
Figure 5.8 demonstrates the training process and the test results of the nRBM; both experiments stabilises earlier than AE training: about 400 and $1,000$ steps respectively. Due to the randomness of the sampling in the nRBM, there are noisy fluctuations in



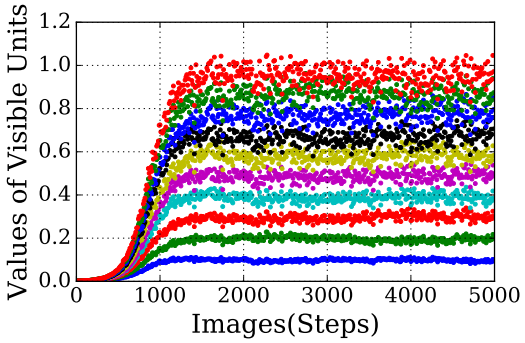
(a) Weights of Exp1



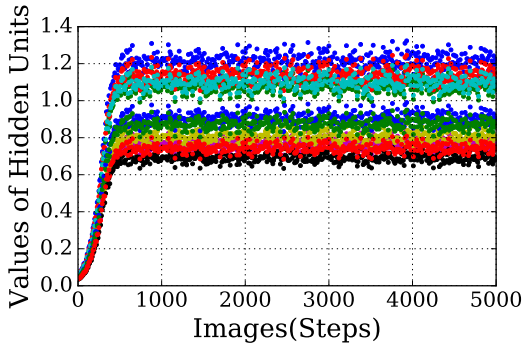
(b) Weights of Exp2



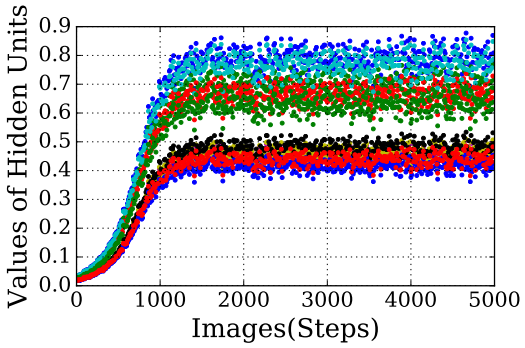
(c) Reconstruction of visible units in Exp1



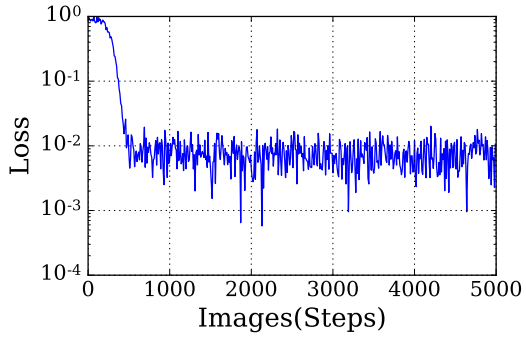
(d) Reconstruction of visible units in Exp2



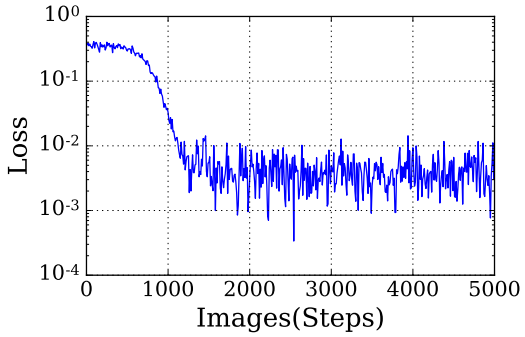
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

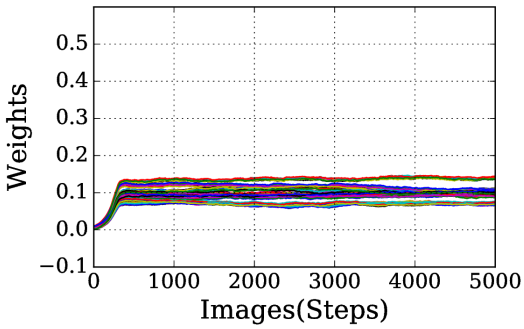


(g) Output of hidden units in Exp1

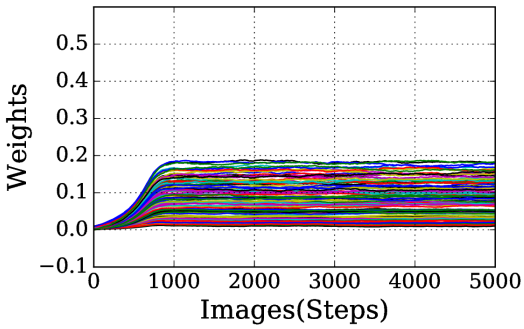


(h) Output of hidden units in Exp2

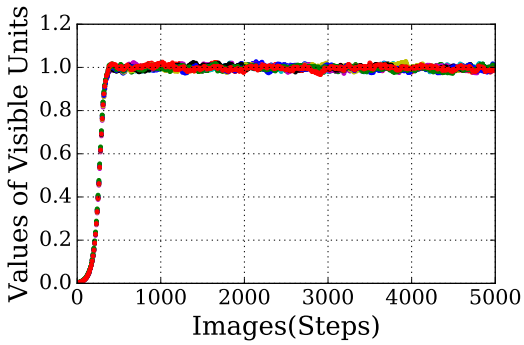
Figure 5.7: Changes of weights, output of visible and hidden units, and mean squared error (loss) during the AE training of the noisy reconstruction tests. Experiments 1) 10 visible units fully connected to 10 hidden units with the count of Poisson spikes firing at 100 Hz which lasted 100 ms; 2) the same network fed with spike count of Poisson spikes at firing rates ranging from 10 Hz to 100 Hz.



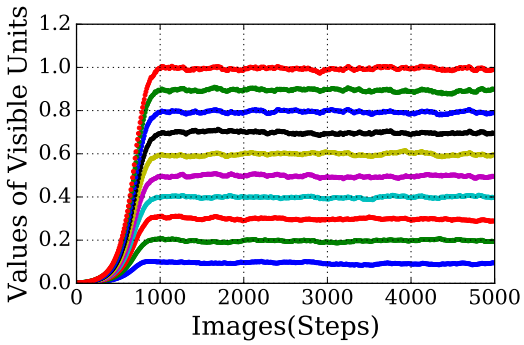
(a) Weights of Exp1



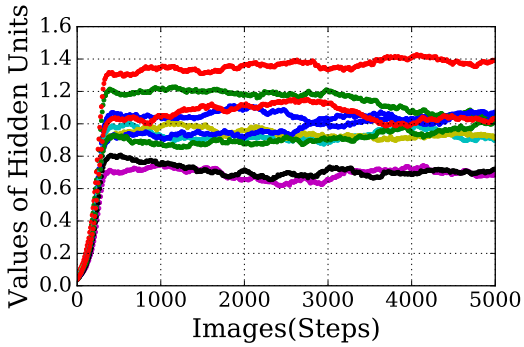
(b) Weights of Exp2



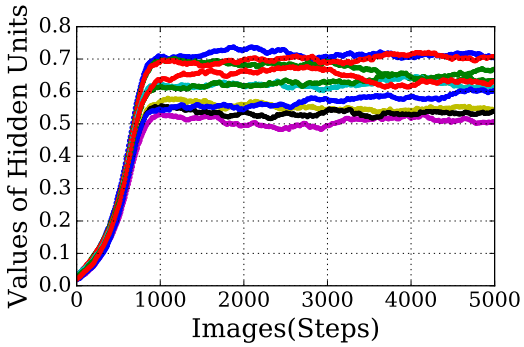
(c) Reconstruction of visible units in Exp1



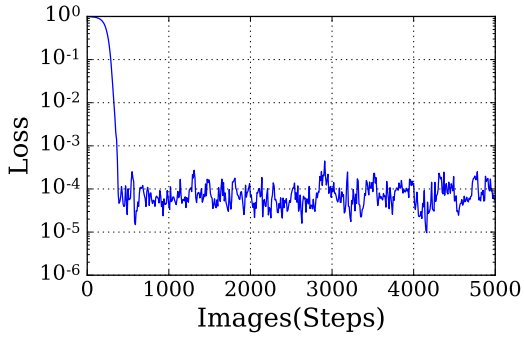
(d) Reconstruction of visible units in Exp2



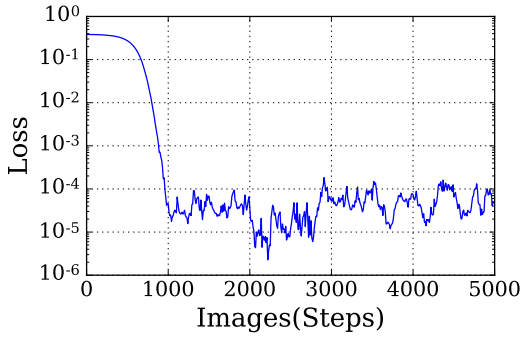
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

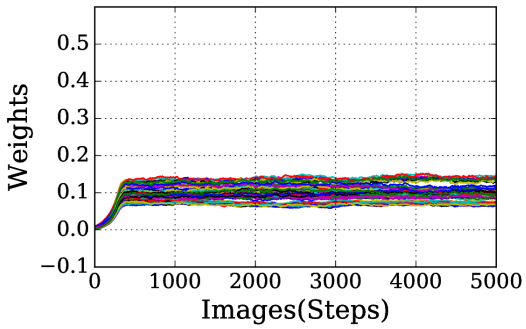


(g) Output of hidden units in Exp1

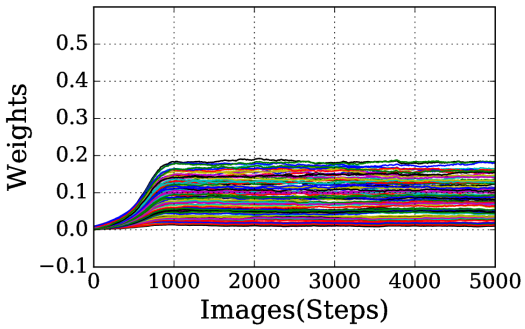


(h) Output of hidden units in Exp2

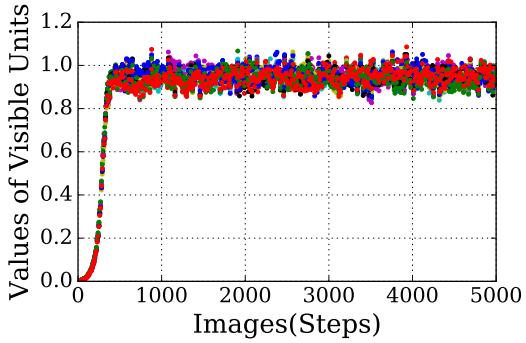
Figure 5.8: Changes of weights, output of visible and hidden units, and mean squared error (loss) during the nRBM training of the reconstruction tests. Experiments 1) 10 visible units fully connected to 10 hidden units with input data of all 1s; 2) the same network fed with 10 values distributed linearly from 0.1 to 1.



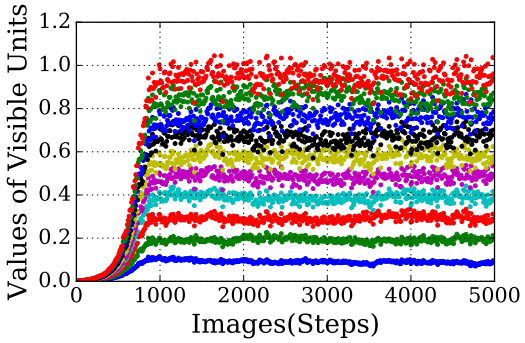
(a) Weights of Exp1



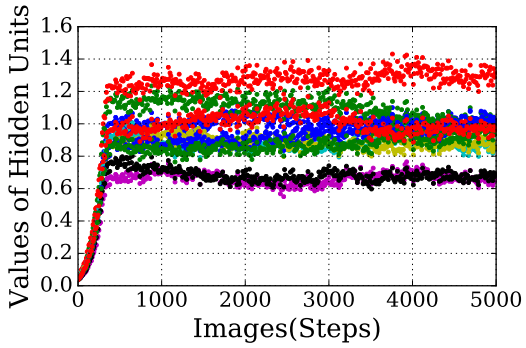
(b) Weights of Exp2



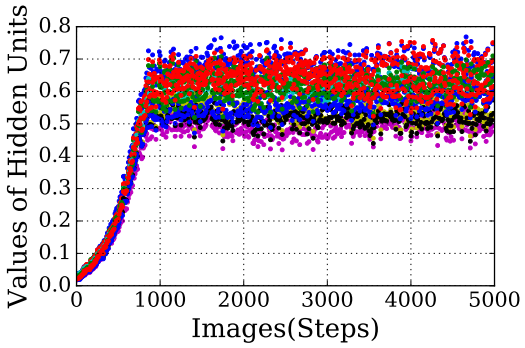
(c) Reconstruction of visible units in Exp1



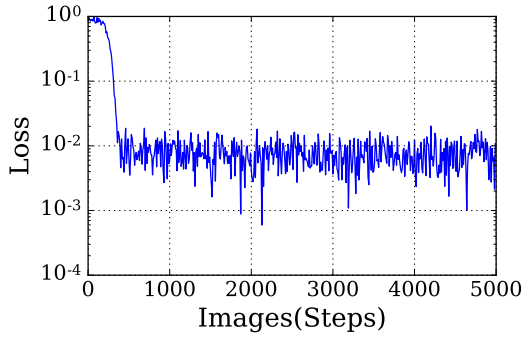
(d) Reconstruction of visible units in Exp2



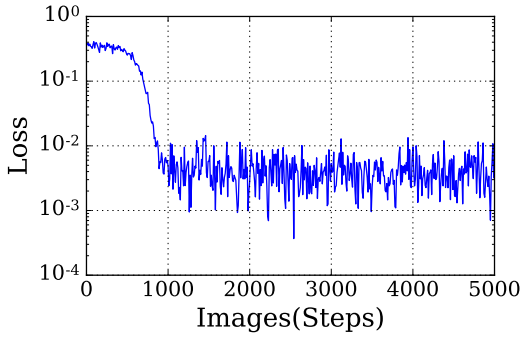
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2



(g) Output of hidden units in Exp1



(h) Output of hidden units in Exp2

Figure 5.9: Changes of weights, output of visible and hidden units, and mean squared error (loss) during the nRBM training of the noisy reconstruction tests. Experiments 1) 10 visible units fully connected to 10 hidden units with the count of Poisson spikes firing at 100 Hz which lasted 100 ms; 2) the same network fed with spike count of Poisson spikes at firing rate ranging from 10 Hz to 100 Hz.

the weight change during training, the reconstructions and the output of hidden units in deterministic testing. In addition, the loss stops declining at around 10^{-4} because of the same reason of randomness.

The same experiments were also carried out with the identical NI, see Figure 5.9. The weight change is slightly noisier than the experiments on clean data, but the noise is more obvious on the output of the visible reconstruction and the hidden units. The same effect of the NI data can be found in Figure 5.7 where the noise in the reconstruction is much reduced compared to the input data and the loss remains about 10^{-2} and $10^{-2.5}$ after the stabilisation, although it takes less time for the nRBM to converge than AE.

5.4.4 Training Spiking AEs

Equation 3.11 states the learning rule for AEs, which can be approximated by adding a positive and a negative STDP in SNN training:

$$\begin{aligned} \Delta w_{ij} &= \eta h_i v_j - \eta h_i v'_j \\ &= STDP(s_{h_i}, s_{v_j}) - STDP(s_{h_i}, s_{v'_j}) \\ \text{where, } \eta_s &= \frac{\eta 10^6}{K^2 \tau_{dur} \tau_{win}} \ . \end{aligned} \tag{5.12}$$

We use simple linear Integrate-and-Fire (IF) neurons to validate the training algorithm, whose membrane potential follows the dynamics:

$$V_i(t+1) = V_i(t) + \sum_j w_{ij} s_j(t) \ , \tag{5.13}$$

and an IF neuron fires when the membrane potential V surpasses the membrane threshold V_{thresh}, and V resets to V_{rest} after firing or when it reduces below V_{rest}. The parameters used are listed in Table 5.1.

The weight increase takes place in the positive STDP learning between the neurons of the input ($\mathbf{v}$) and the hidden units ($\mathbf{h}$); the weight decrease is carried out in the negative STDP learning between the reconstruction ($\mathbf{v}'$) and the hidden neurons ($\mathbf{h}$). Figure 5.4 shows the network architecture of an SAE where the hidden units connect to the reconstruction neurons with the weight matrix $\mathbf{w}$ and the input neurons feed-forward to the hidden layer with the transposed tied weights $\mathbf{w}^{\mathrm{T}}$. The shared weights,

Table 5.1: Parameter setting of SRM and IF neurons for training SAEs and SRBMs.

Parameters	Values	Description
K	100	linear scaling factor
τ_{dur}	100 ms	length of training spike trains
τ_{win}	10 ms	window length of STDP
η	10^{-3}	learning rate of AEs and RBMs
η_{s+}	10^{-4}	positive learning rate of SAEs and SRBMs
η_{s-}	-10^{-4}	negative learning rate of SAEs and SRBMs
V_{rest}	0 mV	resting membrane potential
V_{thresh}	1 mV	membrane threshold

with the three individual populations of neurons, compose the training network of an SAE, see Figure 5.10 (Left).

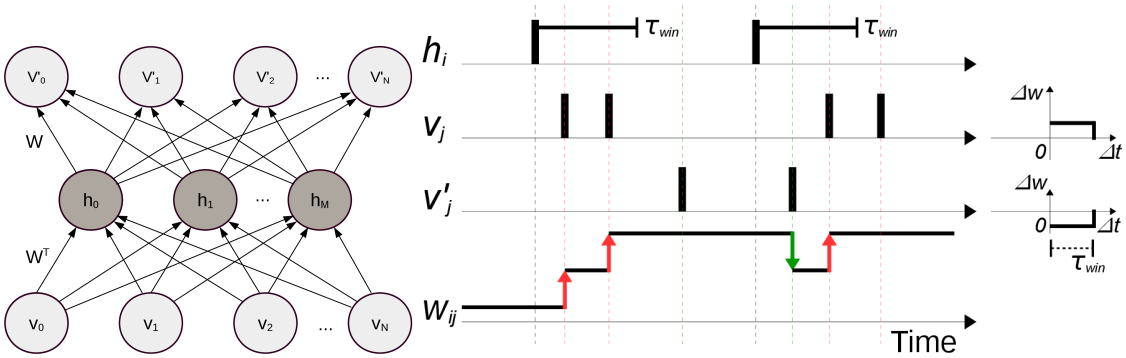
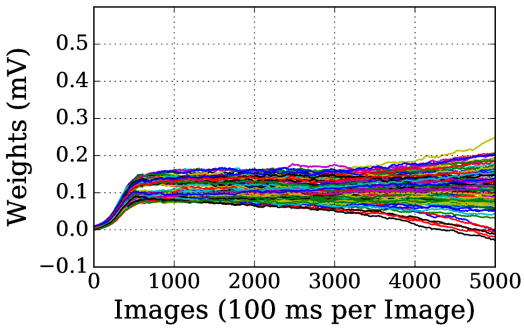


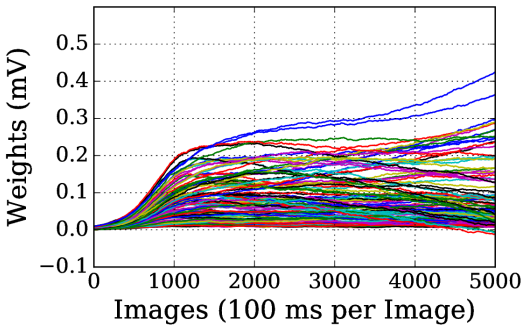
Figure 5.10: Network architecture and the learning algorithm of a spiking AE.

Figure 5.10 (Right) illustrates the learning mechanism of the SAE architecture by giving an example of one hidden neuron h_i connected to an input neuron v_j and connecting to a reconstruction neuron v'_j with the same strength w_{ij}. The weight w_{ij} rises by η_{s+} (marked with an up-arrow) when a spike comes within the period of τ_{win} after the hidden neuron fires; and it decreases by η_{s-} (marked with a down-arrow) if the reconstruction neuron generates a spike in the same time window. If either v_j or v'_j spikes outside the effective STDP window, the weight will remain unchanged. The learning continues as long as the neurons are active, however the weights may stay relatively stable when the input firing rate is the same as the reconstruction's.

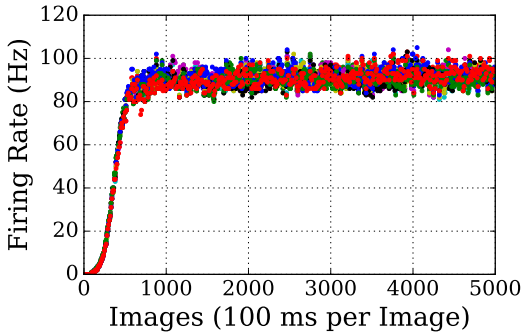
To reproduce the experiments of the AEs, we selected the parameters (see Table 5.1) for training SAEs and also apply the same parameters for SRBM experiments. Each input vector (image) was presented as spiking trains with the length of 100 ms, and the STDP window was set to 10 ms. The input values scaled up by $K = 100$ Hz



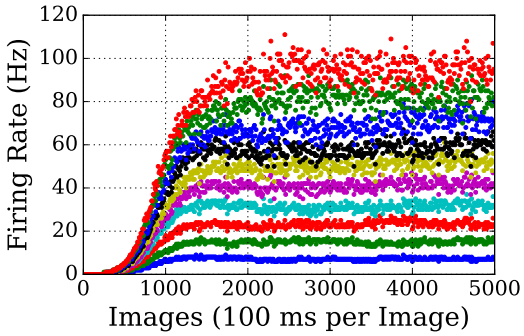
(a) Weights of Exp1



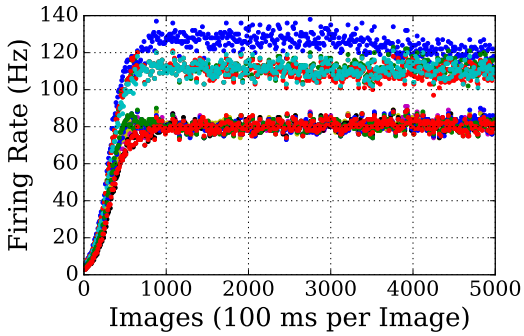
(b) Weights of Exp2



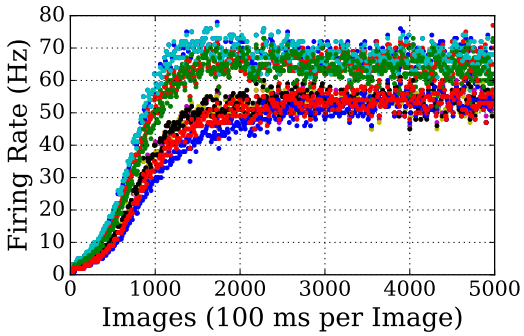
(c) Reconstruction of visible units in Exp1



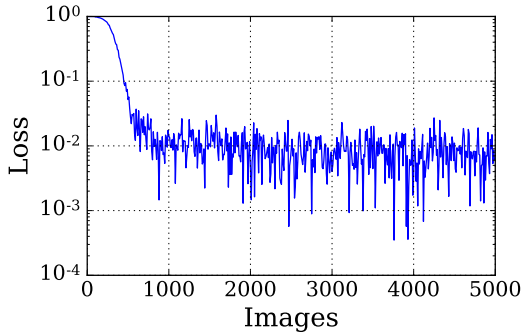
(d) Reconstruction of visible units in Exp2



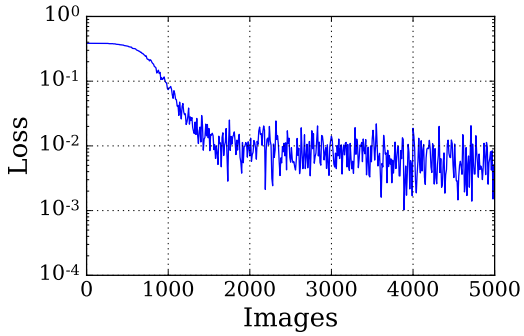
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2



(g) Loss of Exp1



(h) Loss of Exp2

Figure 5.11: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of the spiking AE. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) the same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.

were used as firing rates of the spike trains. Therefore, according to Equation 5.10, the learning rate of SAEs and SRBMs was $\pm 10^{-4}$.

To compare with the AE experiments on NI data, Figure 5.11 records the weight change during training and the results generated by the testing on each image presented for 1 s. The most important and obvious difference is the weight change where the weights do not stabilise but their values diverge during training. The reason for the phenomena will be discussed in Section 5.5. The training is slower than the AE experiments and the loss takes longer to decrease to the same level. This is mainly caused by the low firing rate of both input and hidden units at the beginning of the training. Thus, few, even no, spikes triggered STDP. With the simple input of Exp1, the training performance reaches the same level of 0.01 loss compared to the noisy AE test; however, the SAE does not reconstruct the various input values of Exp2 as accurately as does the AE, since the parameters τ_{win} and τ_{dur} are relatively short. We will present more results conducted with various parameter settings in Section 5.5.

5.4.5 Training Spiking RBMs

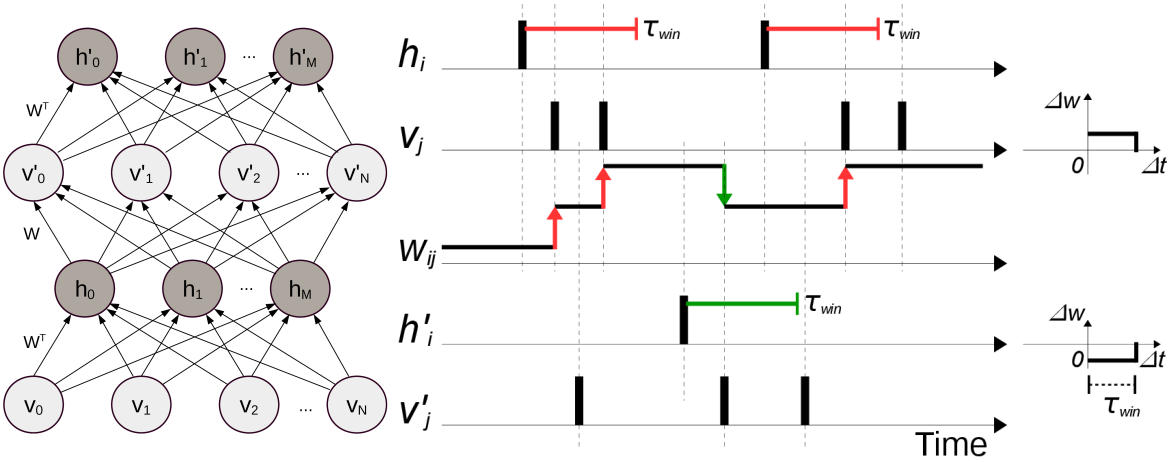
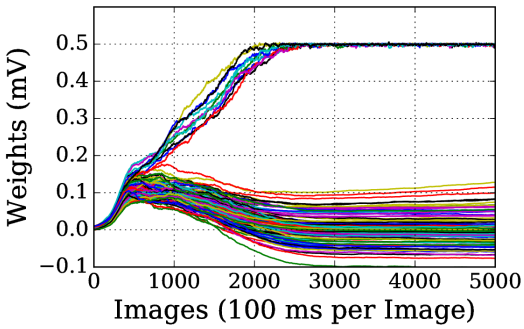
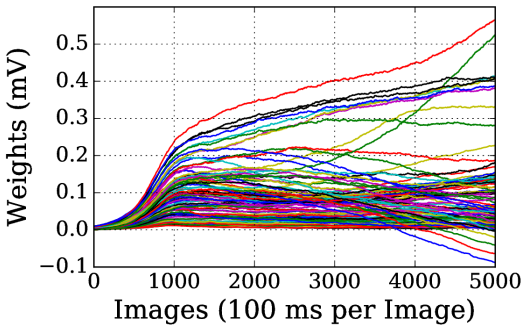


Figure 5.12: Network architecture and the learning algorithm of a spiking RBM.

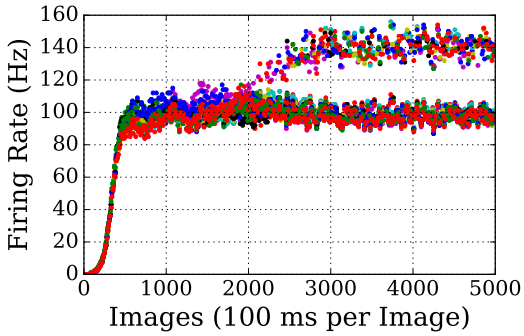
As illustrated in Equation 5.11, the positive weight change is generated from the multiplication of the visible and hidden units where the weight depression comes from the product of the Gibbs sampled hidden and reconstruction values. Figure 5.12 shows the architecture of an SRBM which consists of four layers of neurons: input $\mathbf{v}$, hidden layer $\mathbf{h}$, Gibbs visible (or reconstruction) $\mathbf{v}'$ and Gibss hidden $\mathbf{h}'$ neurons, and the shared weights $\mathbf{W}$.



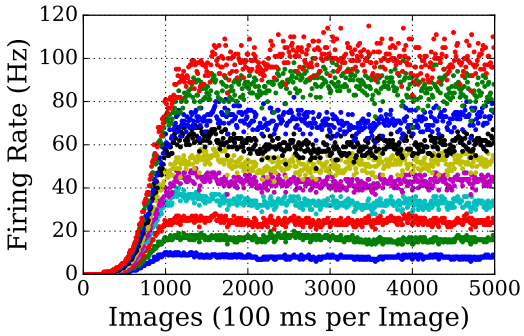
(a) Weights of Exp1



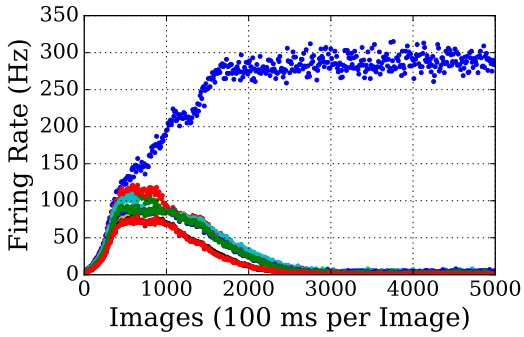
(b) Weights of Exp2



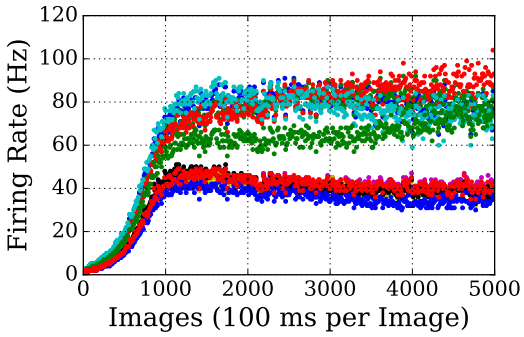
(c) Reconstruction of visible units in Exp1



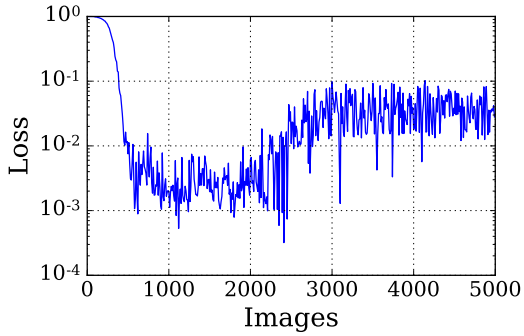
(d) Reconstruction of visible units in Exp2



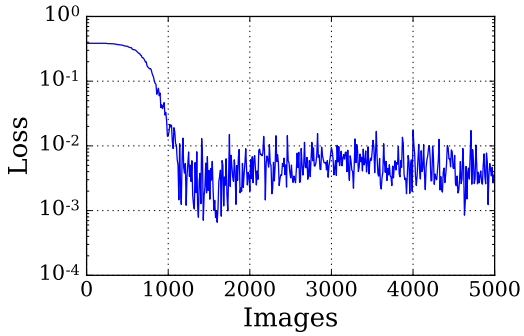
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2



(g) Loss of Exp1



(h) Loss of Exp2

Figure 5.13: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of the spiking RBM. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) the same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.

Figure 5.12 demonstrates the training of an SRBM with a pair of spiking neurons v_j and h_i, the corresponding the Gibbs sampling pair v'_j and h'_i, and the shared weight w_{ij}. Up-arrows mark the increase of the weight when v_j fires during the time window after the spikes of h_i; meanwhile, down-arrows highlight the weight depression when v'_j generates a spike no later than τ_{win} after h'_i fires.

Weight divergence is a more severe problem in the SRBM, see Figure 5.13. Especially for Exp1, only one hidden neuron is active on the later half of the training, thus making the firing rate of the reconstruction jump between two states. Because of the high firing rate of the pre-synaptic neuron, a slight weight increase may generate quite a few spikes for the post-synaptic neuron. Therefore, more balanced activity among hidden neurons can perform better in terms of reconstruction accuracy.

5.5 Problem of Spike Correlations

From the experiments above, the problem of the non-convergence of weight searching appears in training of SAEs and SRBMs using SRM. Instead of stabilising at a (local) minimum of the parameter space, the values of the weights continue to grow or to decay, in fact to diverge, even after the loss locks to a certain level. The diverging weights (see Figures 5.11(a, b) and 5.13(a, b)) not only make the loss increase, for example as shown in Figure 5.13(g), but also lead the weights too far from the expected ~~effective~~ range of the conventional AE or RBM training, refer to Figures 5.7(a, b) and 5.9(a, b).

~~The diverging weight values are caused by correlations between spike trains.~~ Note that the rate multiplication of two spike trains (stated in Equation 5.4) works under the condition of the independence of the spike trains. However, the spike trains generated by a layer of neurons are strongly correlated to the spikes which the lower layer feeds them; and also influence the spike firing of the upper layer. Therefore, any pair of spike trains $\mathbf{v}$ and $\mathbf{h}$, and $\mathbf{h}$ and $\mathbf{v}'$ in training SAEs are correlated, so are the spikes of $\mathbf{h}'$ and $\mathbf{v}'$ of SRBMs. Thus, the unbalanced spike correlations between pairs of spike trains cause the strength of the weights diverge.

Taking SAE training for example, the strength of some synapses continuously increases because they have a relatively strong weight to trigger hidden units to fire,

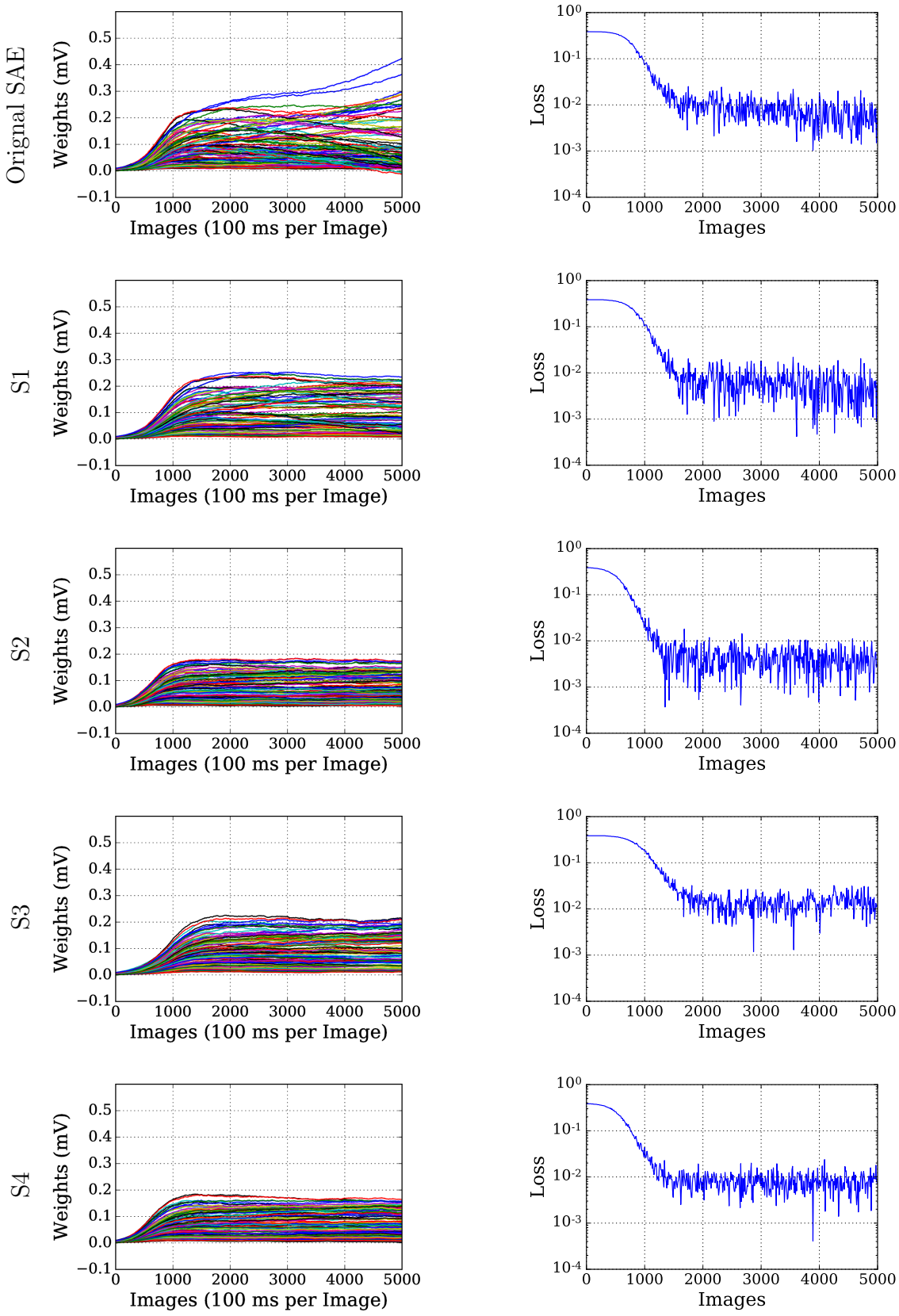
but the hidden units taking their transposed weights have a weaker impact on the firing of the reconstruction neuron. Thus the correlation between v_j and h_i is stronger than between v'_j and h_i, making the positive weight update more frequently than it drops. On the contrary, the decreasing weights appear in the opposite situation when h_i correlates stronger with v'_j than v_j. Regarding the SRBM, the training is even more effected by the correlations where the values of the weights diverge faster.

This section proposes four solutions to reduce the correlation between spike trains, thereby approximating AE and RBM training with equivalent spike-based on-line learning of SAEs and SRBMs. To highlight the performance of the solutions, we apply these methods to the same experiments above, and the default parameters are the same as in Table 5.1.

5.5.1 Solution 1 (S1): Longer STDP Window

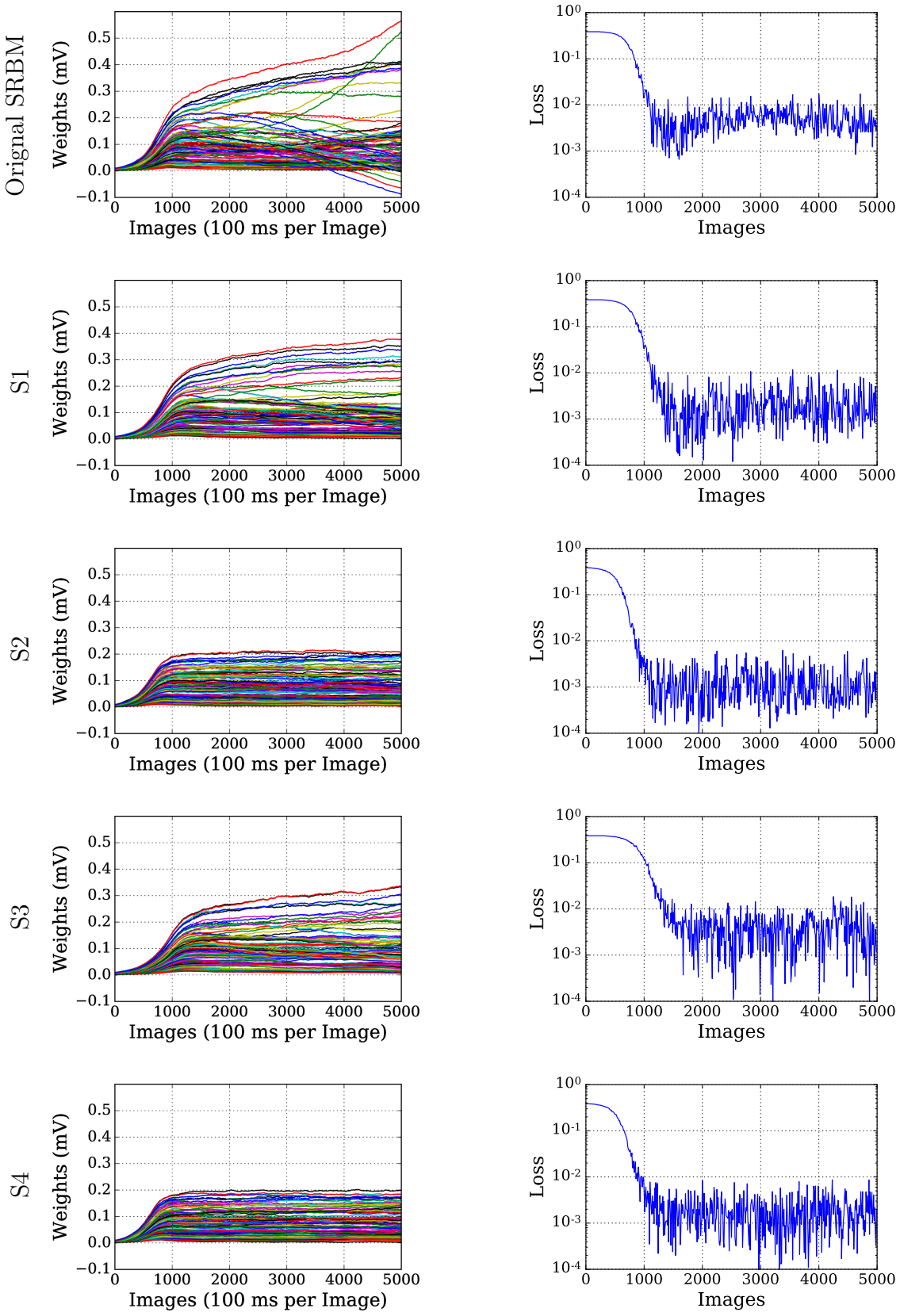
A stronger weight between a pair of pre- and post-synaptic neurons results in a higher probability for the pre-synaptic neuron to trigger a post-synaptic spike in a shorter period; a weaker connection strength usually takes longer to activate a spike. Thus SRM using a short τ_{win} produces a lower rate multiplication than the numerical product, especially for the weak weights. Therefore, we use a longer square STDP window to accumulate more evidence to improve the accuracy of SRM, thus to make SRM more independent of the correlations between spike trains. Of course, an unlimited window length is ideal to eliminate the effect of neural correlations, but a fair trade-off also takes computation and memory use into account. In the experiment, we doubled the window length to 20 ms and set η_s to $\pm 5 \times 10^{-5}$, and recorded the complete results in Figures B.1 and B.2 in Appendix B.

To make the comparisons of all the proposed solutions more straightforward, we list the weights and loss change from Exp2 in Figure 5.14 (SAE) and Figure 5.15 (SRBM). The longer STDP window reduces the weight divergence so that the bidirectional weight change slows down and it kept within a smaller range. In addition, the loss drops to a lower level compared to the original SAE test because the longer STDP window τ_{win} also makes the SRM more accurate which reflects the first property of the SRM algorithm in Section 5.3.



(a) Exp2 Weights (b) Exp2 Loss

Figure 5.14: Comparisons of weights and loss of solutions for training SAE on Exp2: [S1] longer STDP window, [S2] noisy threshold, [S3] teaching signal, and [S4] combined solutions. 10 visible units fully connects to 10 hidden units with Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



(a) Exp2 Weights (b) Exp2 Loss

Figure 5.15: Comparisons of weights and loss of solutions for training SRBM on Exp2: [S1] longer STDP window, [S2] noisy threshold, [S3] teaching signal, and [S4] combined solutions. 10 visible units fully connects to 10 hidden units with Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.

5.5.2 Solution 2 (S2): Noisy Threshold

We introduce a method of 'noisy threshold' (also called escape or hazard model) [Gerstner and Kistler, 2002] to generate noise in the output of spikes. The stochastic threshold of the membrane potential drives the post-synaptic spikes firing in advance of or behind the expected time, thus reducing the correlation between the pre- and post synaptic spikes. The Gaussian noise added to the membrane potential is randomly generated by $\mathcal{N}(0, \sigma)$ mV, where σ is set to 0.2, 20% of $V_{thresh} - V_{rest}$. An appropriate σ is chosen for introducing enough noise to decrease the correlation of the spike trains and also maintain a good loss performance. The complete test results can be found in Appendix B, see Figures B.3 and B.4.

Using the same STDP window τ_{win} of 20 ms, the noisy membrane threshold enhances the decorrelation, thus the weight change remains more subtle than S1, see Figures 5.14(a) and 5.15(a) . Furthermore, the performance of the reconstruction improves in both SAE and SRBM training so that the loss reduces to a lower level and the training accelerates to reach stabilisation.

5.5.3 Solution 3 (S3): Teaching Signal

A Poisson spike train firing at the same rate as the input spikes but generated with a different random seed is independent of the equivalent input spike train and all the spikes generated in the network. We call these spike trains teaching signals ~~s~~ts, because they are only used for weight updates but not conducted into the network. Then the teaching signals are used in ~~$\Delta w_{ij} = STDP(s_j, h_i)$~~ $\Delta w_{ij} = STDP(ts_j, h_i)$ to replace $STDP(v_j, h_i)$ on the positive part of the weight change. Although the method decorrelates the spike trains for weight increase, the negative updates remain influenced by the correlations. Compared to the other solutions, this requires doubled Poisson spike trains: the input spikes work the same to activate the network, and the teaching spikes do not impact on the neural dynamics but only trigger learning on the synapses between the input layer and the hidden layer.

In terms of SAE training, teaching signals alleviate the problem of correlations, as for the noisy threshold, and the variance of the loss decreases (Figure 5.14[S3]). We can observe a higher level of loss compared to the previous solutions, since a constant

negative bias exists in the reconstructions. It is caused by the stronger correlation on the negative weight update, therefore, to weaken the weight decrease the reconstruction had to be lower than expected. The reconstruction bias can be observed in the firing rates of the reconstruction neurons recorded in the complete experimental result in Figures B.3 and B.4 in Appendix B. Regarding the SRBM training (Figure 5.15), the weight divergence also improves compared to S1 but is not as good as S2, since the noisy threshold applies to the entire network whereas the teaching signal works only on the input layer. The average reconstruction loss is also worse than S1 and S2, similar to SAE test, because of the reconstruction bias.

5.5.4 Combined Solutions (S4)

This test combines the solutions of long STDP window, noisy threshold and teaching signals. The results shown in both Figures 5.14 and 5.15 demonstrate the combined effect of the solutions, in that the dynamic trained weights are between the diverging degree of S2 and S3, and the loss level is also kept in between. More detailed experimental results can be found in Figures B.7 and B.8 of Appendices.

All of the above experiments and solutions were also tested on Leaky Integrate-and-Fire (LIF) neurons. The results are in accordance with IF neurons and validate the features of the SRM that the accuracy is independent of the neural model, see Figures B.9 and B.10 in Appendix B. The LIF neuron model follows the dynamics of its membrane potential, which decreases by a constant leak, $l = 0.01$ mV:

$$V_i(t+1) = V_i(t) + \sum_j w_{ij} s_j(t) - l \ . \tag{5.14}$$

~~Results~~

5.6 Case Study: MNIST Test

Having discussed the problem of training SAEs and SRBMs, we finally apply the proposed methods to the MNIST task. Section 5.6.1 describes the network architecture and the training process, and the rest of the section focuses on the performance analysis on the trained weights (Section 5.6.2), classification accuracy (Section 5.6.3), and the reconstruction performance (Section 5.6.4).

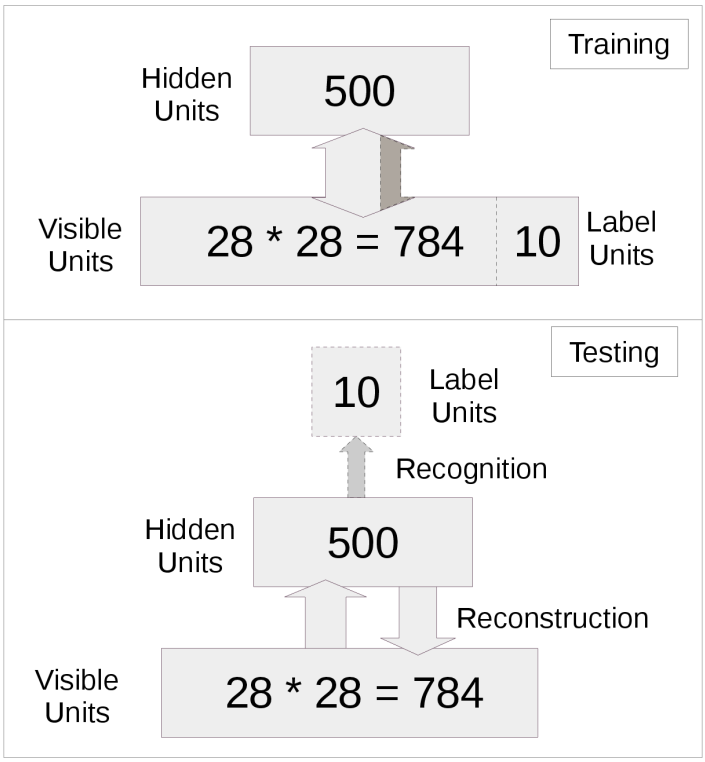


Figure 5.16: AE and RBM structure for MNIST tasks.

5.6.1 Experimental Setup

The training was conducted using a one-layer AE or RBM, see Figure 5.16, which consisted of 794 visible units including 784 neurons representing the images, 10 label units marking the classification, and 500 neurons in the hidden layer. During testing, only the 784 neurons representing image pixels remained as visible units and the other 10 label units worked as the top layer of an ~~MLP~~ Multi-Layer Perceptron (MLP) network for recognising digits. The trained weights of the one-layer AE/RBM were split into the bidirectional weights between visible and hidden units, and the forward connections from the hidden layer to the top layer. The forward path of the network assigned an image to a digit class, and the bidirectional connections reconstructed the input image.

As the baseline for comparison, the same neural network architecture was first tested on conventional AE and nRBM, and then trained on SNNs. Three epochs of all the 60,000 training images were fed into the network in order. In the training of SNNs, as described above in Section 5.4.1, pixel values of images were represented by Poisson spike trains firing at a certain frequency linearly proportional to the original value. The overall count of spikes for each pixel was used as an NI for conventional

training, AE-NI and nRBM-NI, to provide an equal comparison to SNNs. Moreover, all the training images were presented in sequence one by one in an on-line fashion, in other words we used the minimum batch size of 1. In addition, the default parameters of SNNs were the same as listed in Table 5.1 except that in the improved solutions of SNN training the STDP window τ_{win} was doubled to 20 ms and the learning rate η_s was set to 5×10^{-5} accordingly; the initial weights were identical for all the experiments, as were the input spike trains.

5.6.2 Trained Weights

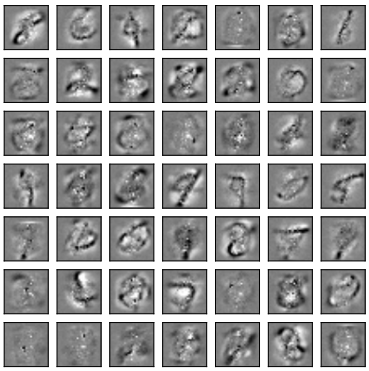
The first comparison is conducted directly on the trained weights after three epochs of the entire training image set, which qualitatively demonstrates the difference of learning performance between conventional methods and the SNN training solutions stated above. We randomly select 49 hidden neurons out of 500 to display the weights between the hidden and visible units (only the 784 neurons representing the images), see Figures 5.17 and 5.18 where the grey scales from white to black represent the values between -0.1 to 0.1. Unlike the diverging weights in the previous section, the trained weights of all the methods stay within a similar range thanks to the large dataset which restricts the weight searching to a certain scope.

The trained weights of AE and AE-NI show little difference. However, the original SAE training results in noisy trained patterns where unexpected white dots appear on the extracted digit features (Figure 5.17(c)). Because of the size limit of the figures, it is not easy to observe the noisy dots, thus we provide a bigger size of these figures in Appendix (Figures B.11 to B.22). The other improved SAE trainings demonstrates a closer feature extraction to the conventional AEs.

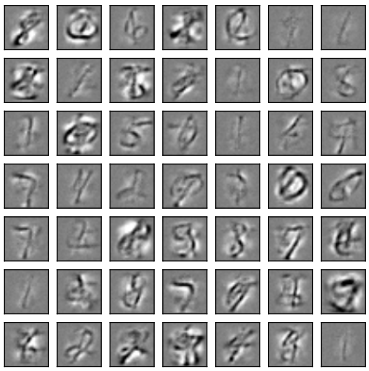
Regarding RBM training, the results in Figure 5.18 show: extracted features of the nRBM consist of continuous strokes although they are not as recognisable as those of the AE, due to the randomness introduced by the noisy sampling of NReLU; the noise in the input data leads to the noisy extracted features where discontinuity of strokes appears; few continuous strokes are generated in the original SRBM training; but more noticeable extracted patterns turn up with the noisy threshold solution of SRBM-S2; patterns generated by SRBM-S3 training show similar continuous strokes as in the original nRBM training; and combined solutions demonstrate mixed patterns

(a) AE

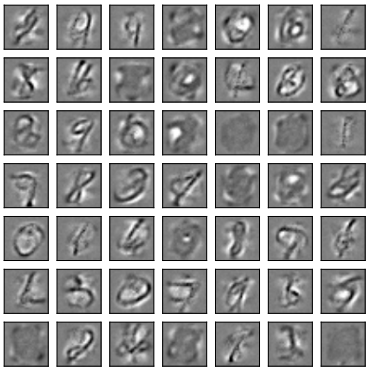
(b) AE-NI



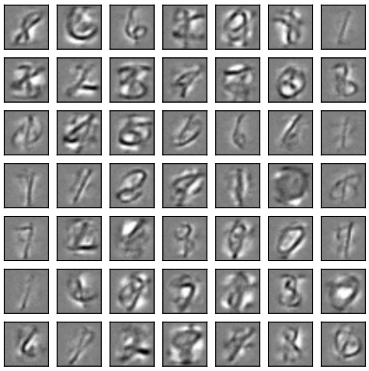
(c) Original SAE



(d) SAE-S2

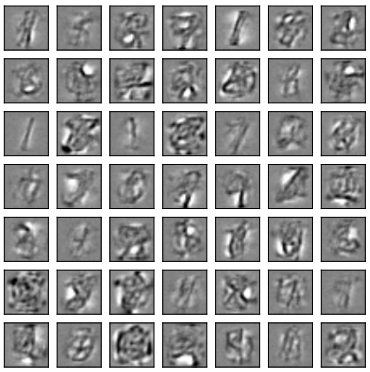


(e) SAE-S3

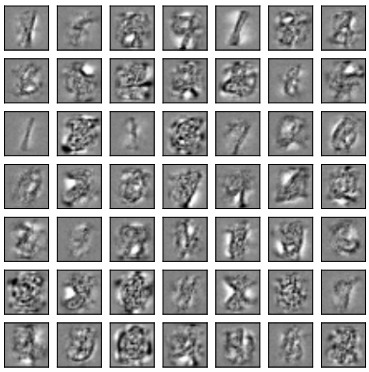


(f) SAE-S4

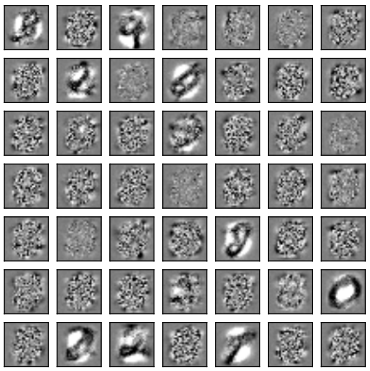
Figure 5.17: Trained weights after 3 epochs of MNIST training using (a) AE, (b) AE-NI, Poisson spike trains as input data, (c) Original SAE, (d) SAE-S2, neurons with noisy thresholds, (e) SAE-S3, extra teaching signal, and (f) SAE-S4, combined solutions of S2 and S3.



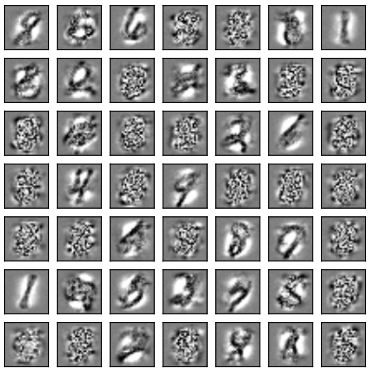
(a) nRBM



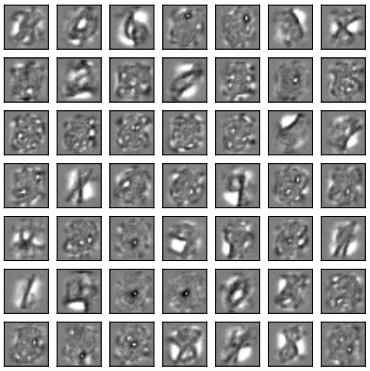
(b) nRBM-NI



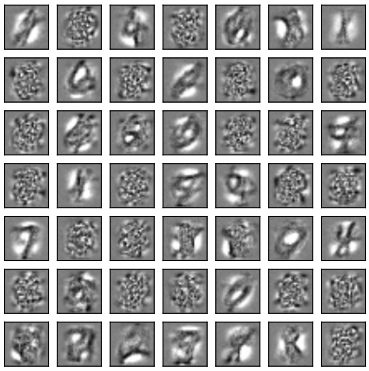
(c) Original SRBM



(d) SRBM-S2



(e) SRBM-S3



(f) SRBM-S4

Figure 5.18: Trained weights after 3 epochs of MNIST training using (a) nRBM, (b) nRBM-NI, Poisson spike trains as input data, (c) Original SRBM, (d) SRBM-S2, neurons with noisy thresholds, (e) SRBM-S3, extra teaching signal, and (f) SRBM-S4, combined solutions of S2 and S3.

of training with S2 and S3 which is closer to the patterns trained with nRBM-NI.

In brief, from the qualitative comparisons on the trained features, the original SAE is the only exception which produce 'substandard' features, noisy reconstructions, among the SAE methods. However, in RBMs, SRBM-S3 generates the most qualitative features of all the SRBM methods and only SRBM-S3 and SRBM-S4 achieves similar learning performance as does the conventional nRBM-NI.

5.6.3 Classification Accuracy

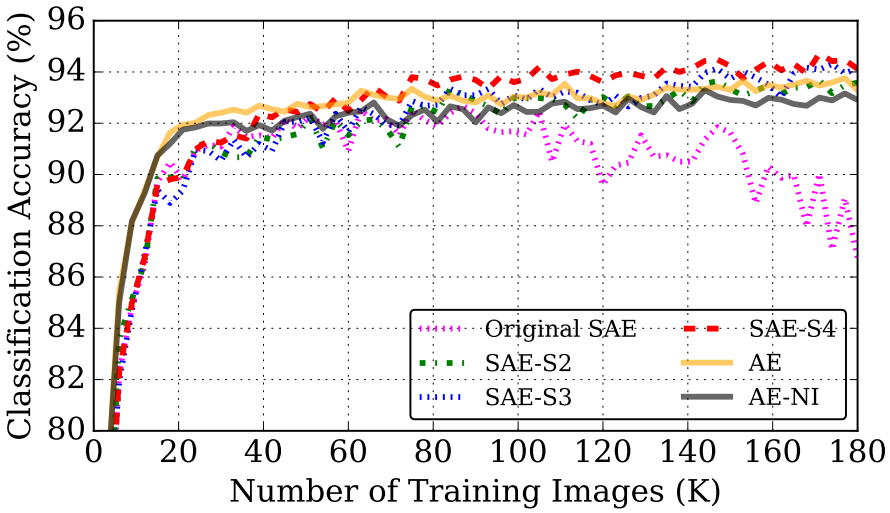
As the most important metric to evaluate the training performance, we test the classification accuracy on SNNs to compare with conventional models. The classification neuron having the highest spike count indicates the predicted class of the given image. A correct recognition only happens when the prediction is the same as the label and only one prediction exists. Thus if two output neurons generate the same number of spikes which is the maximum, still the recognition is failed.

We start from the default configurations of the experiments, then take a closer look into the energy considerations. Thus, there are two parameters to configure the experiments: (1) testing time, which determines the time length an image is presented to the network; and (2) scaling factor K, which controls the input firing rates proportional to the pixel intensities. As the default configuration, the testing time is set to 1 s and K=100 Hz.

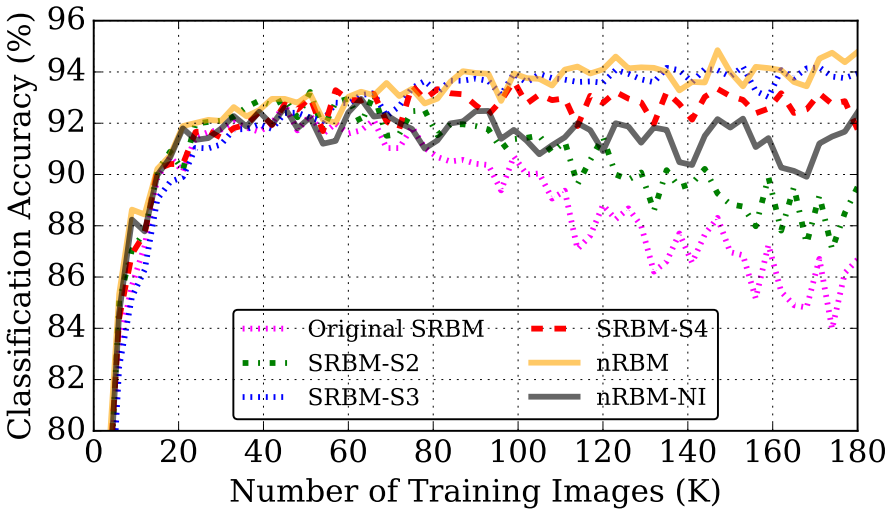
Default configuration

The test with NI on conventional Deep Learning modules is the important baseline for comparisons of conventional ANN training to the equivalent on-line biologically-plausible learning in SNNs. Because, the NI test exploits the same training and test data used in SNNs: the spike count of the Poisson spike trains fed into the SNNs. Thus, the evaluations on Classification Accuracy (CA) quantitatively measures how well the spike-based training fits to the conventional Deep Learning.

The results of conventional and spiking AE are shown in Figure 5.19(a). Most significantly, in accordance with qualitative comparisons in Section 5.6.2, all the SAE training, except for the original SAE, outperforms the AE-NI (93.29%) baseline and achieves equivalent, even better CA than conventional AE with clean data (93.76%).



(a) AEs and SAEs.



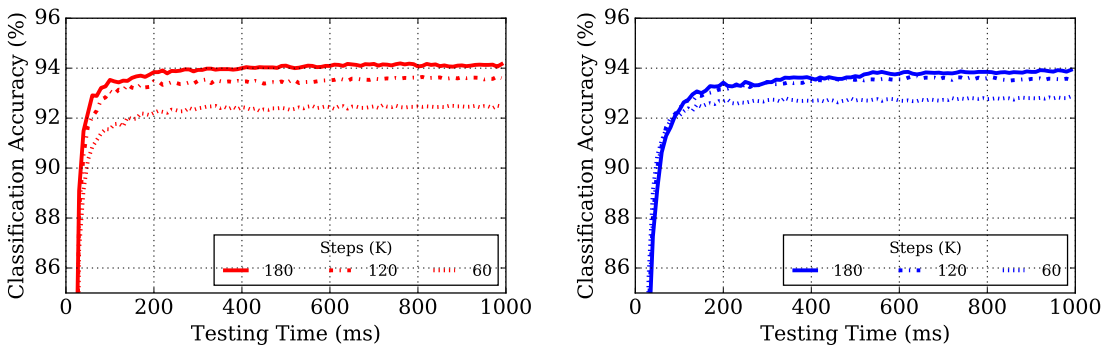
(b) nRBMs and SRBMs.

Figure 5.19: Classification accuracy comparing conventional models with spike-based models. All the training starts with the same initial weights. Every 3K steps of training, the trained weights are recorded and tested on the MNIST testing dataset for classification accuracy. The Poisson spikes trains used for SNN simulations are identical for all the tests.

The training on SAEs is slower than on AE at the beginning, due to low spiking rates (explained in Section 5.4.4). The classification performance of SAEs reaches the AEs' after about 50K training steps, and starts to diverge: the original SAE keeps a CA close to that of AEs until it peaks at 92.71% using 87K steps, and then the CA begins to decay because of the correlation problem; while both SAE-S2 and SAE-S3 have similar performance to AEs and even overtake the best CA of AEs at 93.75% and 94.38% respectively; and the combined solution SAE-S4 performs the best among all the models and achieves 94.72% CA at 171K steps. Thus, decorrelated SAEs are proved to be as competent as AEs in classification tasks and their learning speed closely fits to the conventional AEs.

For the RBM testing, nRBM is more sensitive to the noise of the training images than AE so that the CA of nRBM-NI slowly drops and oscillated more and more strongly as training proceeded, see Figure 5.19(b). The peak performances of nRBM and nRBM-NI appear on 146K and 97K steps at 94.85% and 93.16% respectively. The quantitative test results of SRBMs draw a consistent conclusion with the trained weights described qualitatively above in Section 5.6.2: the original SRBM suffers greatly from drastic performance drop due to the spike correlation; SRBM-S2 performs slightly better since the noisy membrane threshold reduces the spike correlations; SRBM-S3 operates the best which overtakes the nRBM-NI baseline and fits the same learning curve of nRBM; and finally the performance of the combined method is between the S2 and S3 solutions, moreover, exceeds the nRBM-NI baseline. SRBM-S3 achieves the best CA of 94.35% among the SRBM models. In short, although most of the decorrelation solutions performs worse than those of AEs, teaching signals of the S3 solution works extremely well for training SRBM which achieves the same CA and reproduces the learning speed of conventional nRBM.

To compare in the literature, we highlight the advantages of the proposed method. (1) The learning curves of the SNNs are close to the conventional ANNs which proves the accurate estimation of the STDP parameter configurations using SRM; while existing methods [Neil, 2013; Neftci et al., 2013] pick biological settings instead of configurations from mathematical analysis, which results in performance drop comparing to the ANNs. (2) The proposed solutions decorrelate the spike trains and enable the genuine 'live and learn' in on-line SNN training; however most of the published



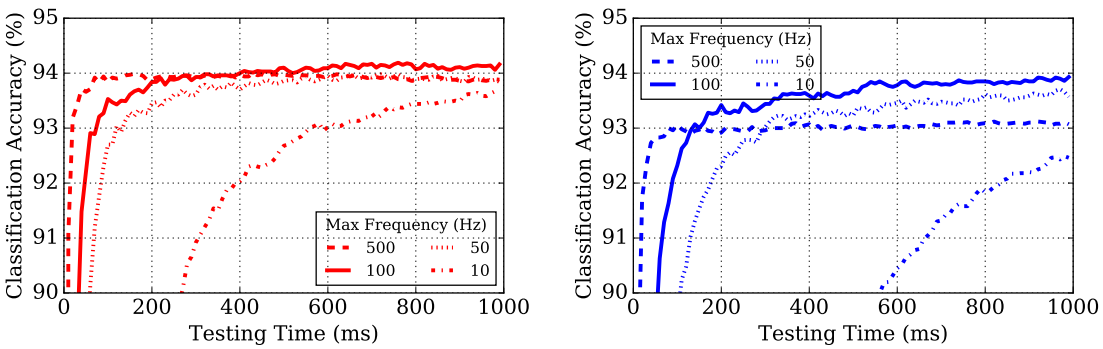
(a) SAE-S4 (b) SRBM-S3

Figure 5.20: Classification accuracy during the entire testing time of 1 s.

papers [Neftci et al., 2013, 2016] only reported the peak learning performance and manually terminated training. (3) SAEs and SRBMs achieve the cognitive capabilities equivalent as the corresponding Deep Learning modules; but most of the methods mentioned above [Neil, 2013; Neftci et al., 2013] experience a performance drop. Although, Neftci et al. [2016] keep the state-of-the art accuracy of 95.6%, they exploited the ~~edge~~ cutting-edge tool, dropout and more training epochs, but did not report the comparison to the corresponding non-spiking RBM-dropout. (4) Our proposed method is supposed to work on any spiking neuron model and requires a simple rectangular STDP, thus easy to be implemented on general neuromorphic hardware; meanwhile others [Neftci et al., 2013, 2016] use specific neuron/synaptic models, extra control signals and certain network architecture, which are difficult to be transferred to hardware platforms.

Testing time

The shorter the period that each image is presented to the network, the lower the energy required when implemented in hardware. Thus to have a closer look at the CA performance over the testing time, we tested the CA at every ms for the entire duration of 1 s, and the best models of SAEs and SRBMs showed the CA in Figure 5.20. The SAE-S4 model can achieve a CA close to their best performance using only 200 ms, which saves 80% of the time and the energy. SRBM-S3 exhibits a continuously improving classification performance, however, as training proceeds it takes longer for the model to converge to its best performance. Hence, about 400 ms is needed for SRBM-S3 to perform close to its best CA.



(a) SAE-S4 (b) SRBM-S3

Figure 5.21: Classification accuracy during the entire testing time period with increasing input spike firing rate.

Spiking rate

Another factor impacting energy use is the mean firing rate of the whole network. If a close classification performance can be achieved with sparse spikes or a lower firing rate, then the neuromorphic implementation will be more energy efficient. Figure 5.21 shows how the firing rates of the input spike trains affect the classification performance over a testing time of 1 s. The scaling factor K of SRM used for training SAEs and SRBMs was set to 100 Hz, thus we took three other values of K for comparison. For SAE-S4, the CA of models using K of 50, 100, 500 converge quickly to a similar value. Thus, for instance, the CA can achieve 93.5% using only half of the firing events in the network and 200 ms for the testing period, which costs 10% of the time and energy use comparing to the default test, but achieves merely 0.5% loss on CA. The CA of the SRBM-S3 is more sensitive to the input rate, where the model with $K = 100$ Hz achieves the highest accuracy. However, similar to SAEs, with half of the events used in the network and 400 ms testing time, the CA is also about 0.5% lower than the model using the preferred input rates.

According to Equation 4.16 the energy use for classifying the whole set of testing images can be estimated more accurately. The mean synaptic event rate for various K values is listed in Table 5.2. Hence, we can estimate their energy cost given that of a single synaptic event of some specific neuromorphic hardware, for example, about 8 nJ on SpiNNaker [Stromatias et al., 2013]. Thus, it may cost SpiNNaker only 0.021 W of power running for $2,000$ s, 42 J, for the whole set of synaptic events for the entire SAE-S4 testing on MNIST. Energy measurement on neuromorphic hardware is beyond

the scope of this Chapter, but will be thoroughly discussed in Chapter 6.

Table 5.2: Mean synaptic event rate of whole network given different scaling factor K.

K	Input(Hz)	SAE-S4(Hz)	SRBM-S2(Hz)
10	1.04×10^3	5.28×10^5	5.34×10^5
50	5.20×10^3	2.63×10^6	2.67×10^6
100	1.04×10^4	5.26×10^6	5.33×10^6
500	5.20×10^4	2.62×10^7	2.65×10^7

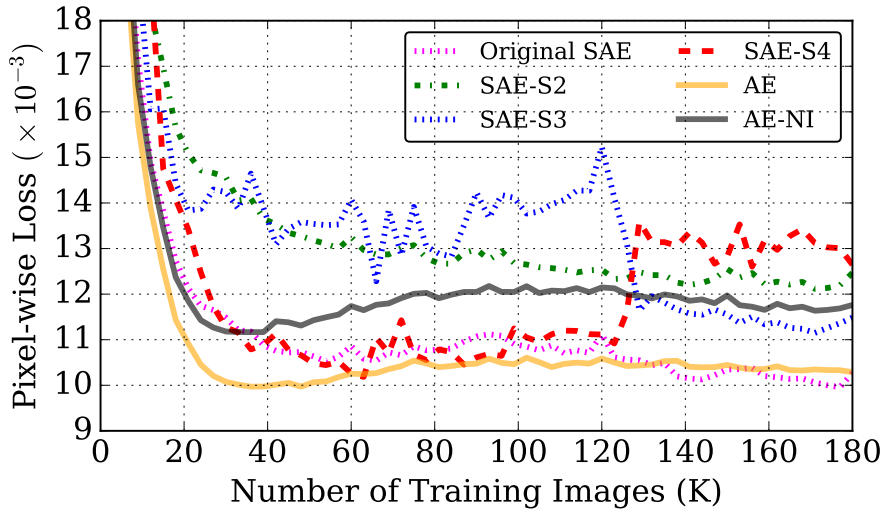
Overall comparisons

In all, SAEs are more suitable for classification tasks than SRBMs. Firstly, the SRBM has an extra layer of Gibbs sampling, thus SAEs require fewer neurons and generate fewer spikes, thus cause less computations and lead to a more energy-efficient implementation on neuromorphic hardware. Secondly, regarding the classification performance, SAE-S4 outperform the conventional AE and all the corresponding SRBM models. Finally, regarding the energy cost in testing only, the SRBM model loses in the comparison due to the use of a longer testing time and a higher synaptic event rate during classification.

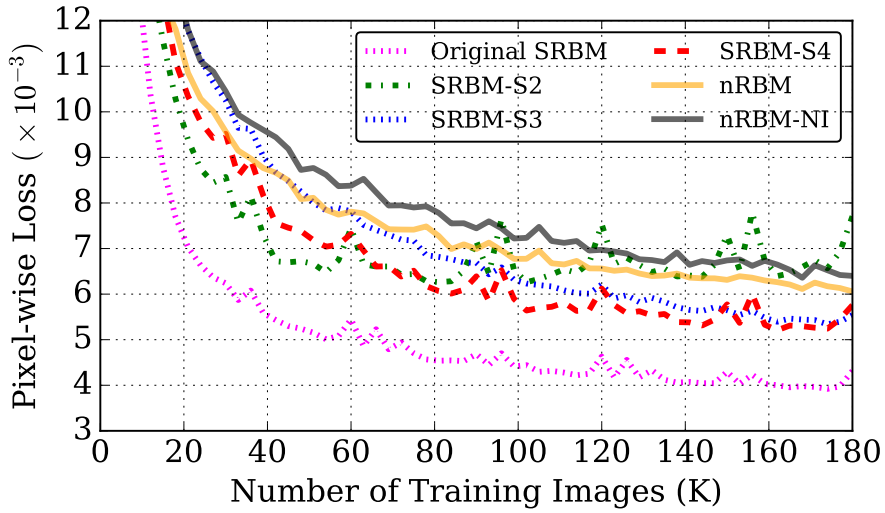
5.6.4 Reconstruction

Reconstruction is another essential function of these unsupervised learning models, such as applications of data de-noising [Xie et al., 2012] and dimension reduction [Hinton et al., 2006]. Figure 5.23 shows how reconstruction improves over increasing training steps for each model. Conventional RBMs demonstrates a better reconstruction capability than AEs in general, see Figure 5.22, thus RBMs are usually used for pre-training AEs or Deep Belief Networks.

It is surprising to see that the original SAE and SRBM trainings perform best in both SAE and SRBM models. A possible explanation could be overfitting. It is easy for the models to reconstruct the images which have been shown, but difficult to generalise the models to unseen data. Classification, here, is equivalent to image in-painting in AEs and RBMs. Therefore, the original SAE and SRBM works well on reconstruction but poorly on classification. Qualitatively shown in Figure 5.23,

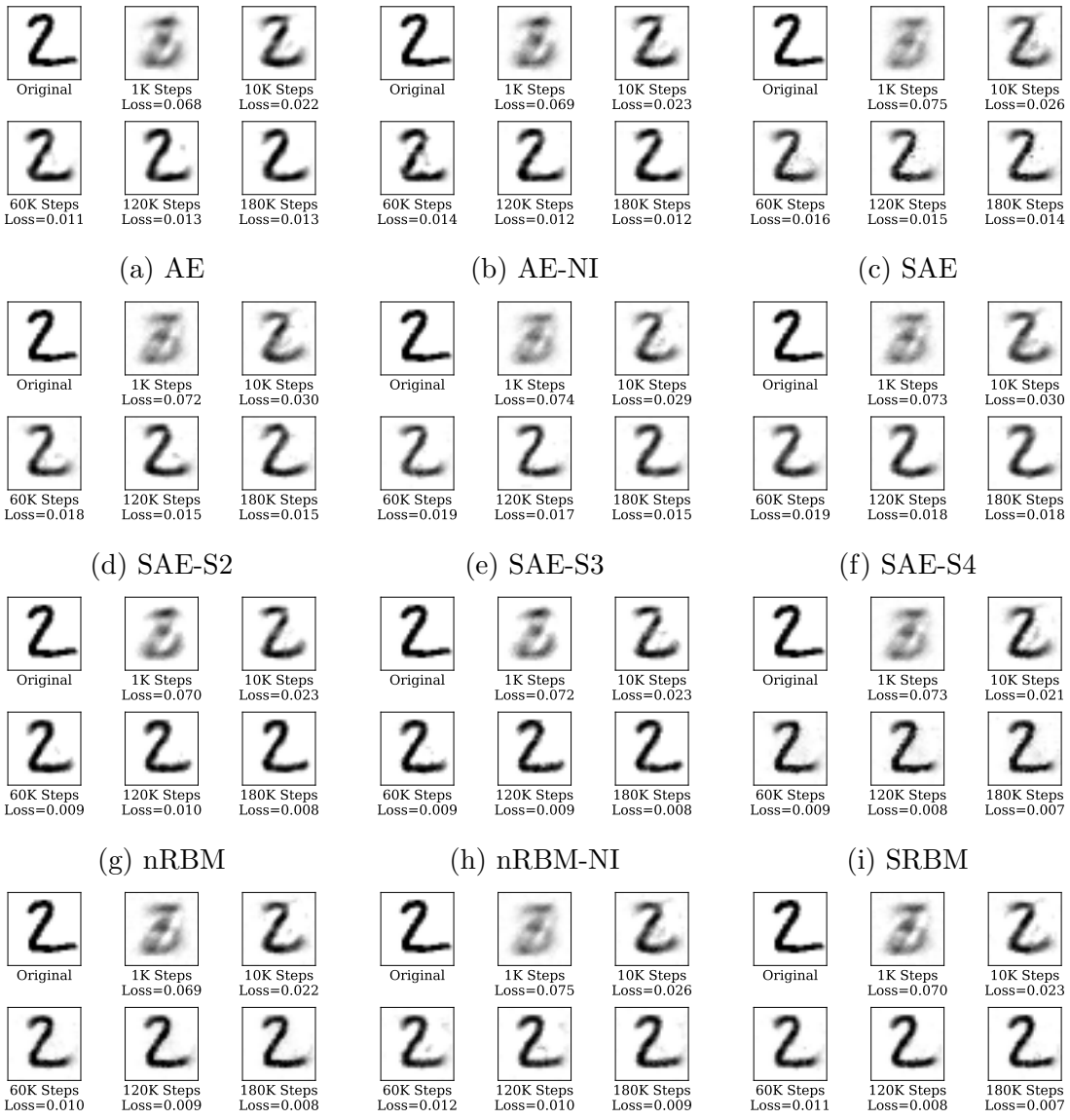


(a) AEs and SAEs.



(b) nRBMs and SRBMs.

Figure 5.22: Loss (mean squared error) pixel-wise of the traditional training method and spike-based models. All the trainings started with same initial weights. Every 3K training steps, the trained weights were recorded and tested on the MNIST testing dataset for reconstruction loss. The Poisson spikes trains used for SNN simulations are identical for all the tests.



(a) AE (b) AE-NI (c) SAE

(d) SAE-S2 (e) SAE-S3 (f) SAE-S4

(g) nRBM (h) nRBM-NI (i) SRBM

(j) SRBM-S2 (k) SRBM-S3 (l) SRBM-S4

Figure 5.23: Reconstruct of the same digit '2' with trained weights using conventional AEs and RBMs to compare with their spike-based models. The Loss (Mean Squared Error) of each reconstruction is labelled under each figure. All the reconstructions are taken on the steps of 1K, 10K, 60K (1 epoch), 120K (2 epochs), and 180K (3 epochs) during training.

SRBMs reconstruct the image closely to the original input, while SAEs reproduce the image more closely to its 'concept', a general digit. Moreover, the original SAE and SRBM, see Figures 5.23(c) and (i), reconstruct the input image more closely than their improved decorrelation methods respectively.

For the other decorrelation solutions in SAE, SAE-S2 converges slowly to the reconstruction loss level of AE-NI, while SAE-S3/S4 are sensitive to the extra teaching signals when trained for reconstruction tasks, thus their performance oscillates roughly during training. In brief, the decorrelated SAE models converge to similar level of reconstruction loss to AE-NI, and original SAE is proved to be equivalent to the conventional AE.

Regarding SRBM training, SRBM-S2 reaches its best loss level the earliest, however vibrates around the same level of conventional nRBM-NI as training continues. Meanwhile, the teaching signal used in SRBM-S3 and SRBM-S4 smooth the loss curve, and both converge to lower levels of loss than the conventional nRBMs.

Therefore, the choice of a proper spiking model for reconstruction depends on the requirements. The original SRBM is outstanding in the reconstruction task alone taking no account of the larger network for training and its poor classification performance. Conversely the SAE-S2 wins in energy-efficient training, effective classification superior than AE, and the stable reconstruction which performs equivalently to AE-NI.

5.7 Summary

This chapter answers the research question: how to train SNNs on-line with biologically-plausible learning rules to catch up with the cognitive capabilities of conventional Deep Learning modules. The proposed SRM method precisely transforms the numerical multiplications, used in conventional unsupervised learning, into parameter configurations of the synaptic plasticity learning rule, STDP, in three easy steps: (1) represent rate multiplication using simultaneous spikes generated from a pair of connected spiking neurons; (2) capture the simultaneous events by the weight change of the synaptic connection using the STDP learning rule; (3) precisely transform the learning rate η to parameters used in SRM, thereby making sure the weight change raised by STDP is equivalent to that of original ANN training. Therefore, the SRM mathematically

estimate the precise parameter configurations of the on-line STDP rules.

The other contribution to the spike-based on-line learning lies in the solutions to decorrelate spike trains, thus to prevent the learning performance from continuous decay after it peaks. It addresses the difficulties of manual stop during training in the existing methods, since the proposed decorrelation solutions enable the on-line SNN learning to be 'live and learn'. Therefore, it paves the way for genuine learning on neuromorphic hardware towards Neuromorphic Cognition.

Experiments show the equivalent learning curves of SAEs and SRBMs to their non-spiking counterparts and the promising results exhibit competent or even superior classification and reconstruction capabilities compared to the conventional Deep Learning modules. Furthermore, in theory our method works on any spiking neuron model and requires a simple rectangular STDP, thus can be easily implemented on general neuromorphic hardware.

Chapter 6

Benchmarking Neuromorphic Vision

The last two chapters have answered the main research question of this thesis showing that Spiking Neural Networks (SNNs) can be trained both on-line and off-line to achieve ~~the equivalent cognitive capabilities of~~ equivalent recognition performance of the Artificial Neural Networks (ANNs). To provide meaningful comparisons between ~~theses~~ these proposed SNN models and other existing methods within this rapidly advancing field of Neuromorphic Engineering (NE), we propose that a large dataset of ~~spike-based visual stimuli~~ spike-encoded images/videos is needed and a corresponding evaluation methodology is also required to estimate the overall performance of SNN models and their hardware implementations.

6.1 Introduction

Today, increasing attention is being paid to research into spike-based neural computation both to gain a better understanding of the brain and to explore biologically-inspired computation. Within this field, the primate visual pathway and its hierarchical organisation have been extensively studied. SNNs have been successfully applied to visual recognition and classification tasks. In addition, Neuromorphic hardware have enabled large-scale SNNs to run in (or even faster than) real time, and also made spike-based neural vision processing accessible on mobile robots. Besides, neuromorphic sensors such as silicon retinas are able to feed such mobile systems with real-time

visual stimuli.

A new set of vision benchmarks for spike-based neural processing are now needed to measure progress quantitatively within this rapidly advancing field. In this chapter we first propose an initial NE dataset based on standard computer vision benchmarks and that uses digits from the MNIST database. This dataset is compatible with the state of current research on spike-based image recognition. The corresponding spike trains are produced using a range of techniques: rate-based Poisson spike generation, rank order encoding, and recorded output from a silicon retina with both flashing and oscillating input stimuli. In addition, a complementary evaluation methodology is presented to assess both model-level and hardware-level performance. Finally, we demonstrate the use of the dataset and the evaluation methodology using an SNN model to validate the performance of the models and their hardware implementations.

With this dataset we hope to (1) promote meaningful comparisons between algorithms and neuromorphic platforms in the field of NE, (2) allow comparison with conventional non-spiking methods on image recognition, (3) provide an assessment of the state of the art in spike-based visual recognition, and (4) help researchers identify future directions and advance the field.

In Section 6.2, diverse neuromorphic vision tasks are introduced to illustrate the requirements for a unified spike-based dataset. The rest of this chapter is structured as follows: Section 6.3 elaborates the purpose and protocols of the proposed dataset and describes the sub-datasets and the methods employed to generate them; Section 6.4 demonstrates the suggested evaluation methodology for use with the dataset. Section 6.5 presents an SNN case study as demonstrations of using the dataset to assess model performance and benchmark hardware platforms. Finally, Section 6.6 summarises the chapter.

6.2 Related Work

Researchers are using the capabilities created by rapid developments in neuromorphic engineering to address the dual aims of understanding brain functions and building

brain-like machines [Furber and Temple, 2007]. Neuromorphic engineering has delivered biologically-inspired sensors such as DVS (Dynamic Vision Sensor) silicon retinas [Serrano-Gotarredona and Linares-Barranco, 2013; Delbruck, 2008; Yang et al., 2015; Posch et al., 2014], which offer the prospect of low-cost visual processing thanks to their event-driven and redundancy-reducing style of information representation. Moreover, SNN simulation tools [Davison et al., 2008; Gewaltig and Diesmann, 2007; Goodman and Brette, 2008] and neuromorphic hardware platforms [Furber et al., 2014; Schemmel et al., 2010; Benjamin et al., 2014; Merolla et al., 2014] have been developed to allow exploration of the brain by mimicking its functions and developing large-scale practical applications [Eliasmith et al., 2012]. Achieving the brain's energy efficiency motivates the development of neuromorphic hardware, since the human brain has a power consumption of only about 20 W [Drubach, 2000]. In the case of visual processing, the brain can accurately recognise objects remarkably quickly, e.g. in 200 ms in monkeys [Fabre-Thorpe et al., 1998], even with short presentations (less than 100 ms) of the target objects [Keysers et al., 2001]. Such rapid and highly accurate recognition is the target of modelling spike-based visual recognition.

Inspired by biological studies of the visual ventral pathway, SNN models have successfully been adapted to visual recognition. Riesenhuber and Poggio [1999] proposed a quantitative modelling framework for object recognition with position-, scale- and view-invariance. Their cortex-like model has been analysed on several datasets [Serre et al., 2007]. Recently Fu et al. [2012] reported that their SNN implementation was capable of recognising facial expressions with a Classification Accuracy (CA) of 97.35% on the JAFFE dataset [Lyons et al., 1998] which contains 213 images of 7 facial expressions posed by 10 individuals. According to Van Rullen and Thorpe [2002], the first wave of spikes carry explicit information through the ventral stream and in each stage meaningful information is extracted and spikes are regenerated. Therefore, using one spike per neuron, similar to the first spiking wave in biology, Delorme and Thorpe [2001] reported 100% and 97.5% accuracies on the face identification task over training (40 individuals $\times$ 8 images) and testing data (40 individuals $\times$ 2 images).

Convolutional Neural Networks (ConvNets), inspired by the receptive fields of the visual cortex, have been implemented on spiking neurons. An early Spiking ConvNet

model identified the faces of 35 persons with a CA of 98.3% exploiting simple integrate and fire neurons [Matsugu et al., 2002]. Another Spiking ConvNet model [Zhao et al., 2015] was trained and tested both with DVS raw data and Leaky Integrate-and-Fire (LIF) neurons. It was capable of recognising three moving postures with a CA of about 99.48% and classifying hand-written digits with 88.14% accuracy on the MNIST-DVS dataset (see Section 6.3.2). In a further step forward, Camunas-Mesa et al. [2012] implemented a convolution processor module in hardware which could be combined with a DVS for high-speed recognition tasks. The inputs of the ConvNet were continuous spike events instead of static images or frame-based videos. The chip was capable of detecting the four suits in a 52-card deck which was browsed rapidly in only 410 ms. Similarly, a real-time gesture recognition model [Liu and Furber, 2015] was implemented by us on a neuromorphic system with a DVS as a front-end and a SpiNNaker [Furber et al., 2014] machine as the back-end, where LIF neurons built up the ConvNet configured with biological parameters. In this study's largest configuration, a network of 74,210 neurons and 15,216,512 synapses used 290 SpiNNaker cores in parallel and reached 93.0% accuracy.

Spike-Timing-Dependent Plasticity (STDP) as a learning mechanism based on biological observations has been applied to vision tasks. As mentioned in Chapter 1, these models typically comprise only two neural layers and exploit STDP and/or Winner-Take-All (WTA) circuits on the synaptic connections. Bichler et al. [2012] demonstrated an unsupervised STDP learning model to classify car trajectories captured with a DVS retina. A similar model trained with STDP using a ~~winner-take-all (WTA)~~ WTA circuit was tested on a Poissonian spike presentation of the MNIST dataset achieving a performance of 95.0% [Diehl et al., 2015a]. Another STDP trained model tested on computer simulation reached a 93.3% CA on MNIST and had the potential to be implemented using memristors [Bill and Legenstein, 2014]. Adding synaptic internal state enabled STDP to work on bistable synapses, and this method achieved the best performance of 96.5% among similar network architectures [Brader et al., 2007].

Deep Learning have exceeded human-level performance on image classification tasks [He et al., 2015], but mainstream Deep Learning research is focussed on continuous rather than spiking neural networks. The spiking deep network has great potential to combine remarkable performance with energy-efficient training and operation. Early

research into spiking deep networks focussed on converting off-line trained deep network into SNNs [O'Connor et al., 2013]. The network was initially implemented on an FPGA and achieved a CA of 92.0% [Neil and Liu, 2014], while a later implementation on SpiNNaker scored 95.0% [Stromatias et al., 2015a]. Recent advances have contributed to better translation by using modified units in a ConvNet [Cao et al., 2015] and tuning the weights and thresholds [Diehl et al., 2015b]. The latter paper claims a state-of-the-art performance (99.1% on the MNIST dataset) compared to the original ConvNet. The current trend towards training deep SNNs on-line using biologically-plausible learning methods is also promising. An event-driven Contrastive Divergence (CD) training algorithm for Restricted Boltzmann Machines (RBMs) was proposed for Deep Belief Networks (DBNs) using LIF neurons with STDP synapses and verified on MNIST with a CA of 91.9% [Neftci et al., 2013]. The work extended to use stochastic synapses, and the recognition performance increased to 95.8% [Neftci et al., 2016]. More related work and detailed description of current research on deep SNNs can be found in Chapter 4 and Chapter 5.

Despite the promising research on SNN-based vision recognition, there is no commonly used database in the format of spike stimuli. In the studies listed above, all of the vision data used are in one of the following formats: (1) raw grey-scale images data; (2) pixel-intensity-driven rate-based Poisson spike trains; (3) unpublished spike-based videos recorded from DVS silicon retinas. However, in the field of conventional non-spiking computer vision, there are a number of datasets playing important roles at different times and with various objectives [LeCun et al., 1998; Deng et al., 2009; Blank et al., 2005; Liu et al., 2009]. In consequence, a new set of spike-based vision datasets is now needed to quantitatively measure progress within the rapidly advancing field of spike-based visual recognition and to provide resources to support objective competition between researchers.

Apart from using spikes instead of the frame-based data used in conventional computer vision, new concerns arise when evaluating neuromorphic vision, such as latency and energy consumption, in addition to recognition accuracy. These concerns naturally derive from the goal of spike-based visual recognition: mimicking the fast recognition with low-energy processing in the brain. Therefore a set of common metrics for performance evaluation in spike-based vision is also required to assess SNN models and their

hardware implementations. In this chapter we propose a large dataset of spike-based visual stimuli and a complementary evaluation methodology. Just as research in this field is an expanding and evolving activity, the dataset will be adapted and extended to fit new requirements presented by advances in the field.

6.3 NE Dataset

6.3.1 Guiding Principles

The NE database we propose here is a developing and evolving dataset~~consisting of~~. It contains various spike-based representations of images~~and videos~~/videos and will also expand to include increasing numbers of encoding methods and commonly used datasets of Computer Vision. The spikes are either generated from spike encoding methods which convert images or frames of videos into spike trains, or recorded from DVS silicon retinas. The spike trains are in the format of Address-Event Representation (AER) [Mahowald, 1992] data, which are suitable for both event-driven computer simulations and neuromorphic systems. AER was originally proposed as a time-multiplexed spike communication protocol where each time a neuron produces a spike an event is generated that codes the spiking neuron's address on a fast time-multiplexed digital bus. The recorded AER data consists of a list of events, each one containing the time stamp of a spike and the address of the neuron which generated the spike. With the NE dataset we hope:

- *to promote meaningful comparisons between algorithms and neuromorphic platforms in the field of NE.* The NE dataset provides a unified format of AER data to meet the demands of spike-based visual stimuli. It also encourages researchers to publish and contribute their data to build up the NE dataset.

- *to allow comparison with conventional non-spiking methods on image recognition.* We expect the dataset to support this comparison using spiking versions of existing vision datasets. Thus, conversion methods are required to transform datasets of images and frame-based videos into spike stimuli. More biologically-accurate and better information preserving schemes are welcome.

- *to provide an assessment of the state of the art in spike-based visual recognition.* To reveal the accuracy, speed, and energy-efficient recognition of neuromorphic approaches, we need not only a spike-based dataset but also an appropriate evaluation methodology. The evaluation methodology will be constantly improving along with the evolution of the dataset.

- *to help researchers identify future directions and advance the field.* The development of the dataset and its evaluation methodology will introduce new challenges for the ~~neuromorphic engineering~~ NE community. However, these must represent an appropriate degree of difficulty: a too-easily-solved problem turns into a tuning competition, while a problem that is too difficult will not yield meaningful assessment. So suitable problems should continuously be added to promote future research.

6.3.2 The Dataset: NE15-MNIST

The first proposed dataset in the benchmarking system is NE15-MNIST (Neuromorphic Engineering 2015 on MNIST). NE15-MNIST is the spiking version of an original non-spiking dataset which was downloaded from the MNIST Database of Handwritten Digits [LeCun et al., 1998] website[1]. Due to its straightforward target of classifying real-world images, the plain format of the binary data and simple patterns, MNIST has been one of the most popular datasets in computer vision for over 20 years. MNIST is a popular task among the neuromorphic vision research community. The converted MNIST dataset consists of four subsets which were generated for different purposes:

- *Poissonian*, which encodes each pixel as a Poisson spike train and is intended for benchmarking existing rate-based SNN models.

- *FoCal (Filter Overlap Correction ALgorithm)*, to promote the study of spatio-temporal algorithms applied to recognition tasks using small numbers of input spikes. This subset is built by my colleague Garibaldi Pineda-García.

- *DVS recorded flashing input*, to encourage research into fast recognition methods to mimic the rapid and accurate 'core recognition' in the primate ventral visual

[1] `http://yann.lecun.com/exdb/mnist/`

pathway [DiCarlo et al., 2012].

- *DVS recorded moving input*, to trigger the study of algorithms targeting continuous input from real-world sensors for implementation, for example, on mobile neuromorphic robots. This subset is contributed by our cooperator Teresa Serrano-Gotarredona.

The dataset is published in the GitHub: `https://github.com/NEvision/NE15`.

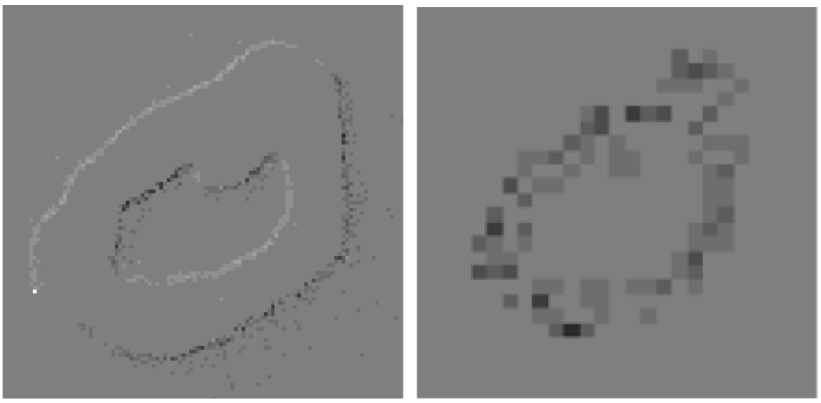
6.3.3 Data Description

Two file formats are supported in the dataset: the jAER format [Delbruck, 2008] (.dat or .aedat), and binary files in NumPy [Walt et al., 2011] (.npy) format. The spikes in jAER format, whether recorded from a DVS retina or artificially generated, can be displayed by the jAER software. Figure 6.1(a) is a snapshot of the software displaying a .aedat file which was recorded from a DVS retina [Serrano-Gotarredona and Linares-Barranco, 2013]. The resolution of the DVS recorded data is 128×128. The second spike-based format used is a list of spike source arrays in PyNN [Davison et al., 2008], a description language for building spiking neuronal network models. Python code is provided for converting from either file format to the other. The duration of the artificially-generated data can be configured using the Python code provided, while the recorded data varies in duration: 1 s for the flashing input, and 3.2 to 3.4 s for the moving input.

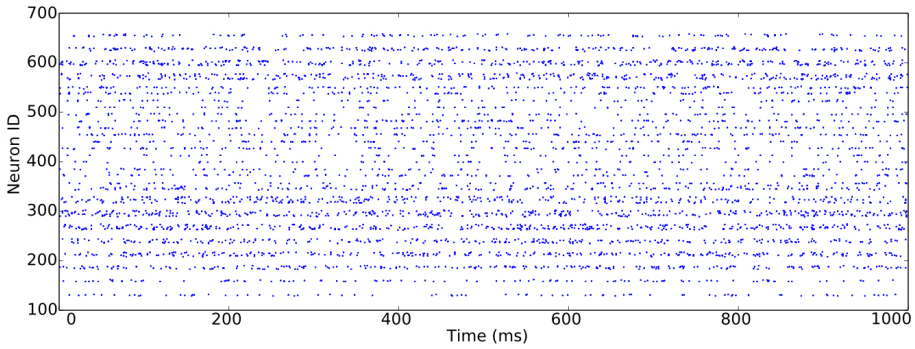
Poissonian

The timing of spikes in the cortex is highly irregular [Squire and Kosslyn, 1998]. An interpretation is that the inter-spike interval reflects a random process driven by the instantaneous firing rate. If the generation of each spike is assumed to be independent of all other spikes, the spike train is seen as a Poisson process. The spike rate can be estimated by averaging the pooled responses of the neurons.

As stated above, rate coding is generally used in presenting images as spike trains. The spike rate of each neuron accords with the intensity of the corresponding pixel. Instead of providing exact spike arrays, we share the Python code for generating the spikes. Each recognition system may require different spike rates and durations. The



(a) DVS recording. (b) Poisson spike trains.



(c) The raster plot of the Poisson spike trains.

Figure 6.1: Snapshots of the jAER software displaying spike-encoded videos. The same image of digit '0' is transformed into spikes by DVS recording and Poisson generation, where (a) ~~DVS recording~~ and (b) ~~Poisson generation~~show the snapshots of the jAER software. They visualise the spike counts for each pixel of an image. (c) A raster plot of the Poisson spike trains.

generated Poisson spike trains can be in both jAER and PyNN spike source array formats. Thus, it is easy to visualise the digits and also to couple the spike trains into spiking neural networks. Because different simulators generate random Poisson spike trains with different mechanisms, languages and codes, using the same dataset enables performance evaluation on different simulators without the confusion created by differences in input. The same digit displayed in Figure 6.1(a) can be converted into Poisson spike trains, see Figure 6.1(b). A raster plot of the Poisson spike trains is shown in Figure 6.1(c).

Rank Order Encoding

A different way of encoding spikes is to use a rank order code; this means keeping just the order in which the spikes fired and disregarding their exact timing. Rank-ordered spike trains have been used in vision tasks under a biological plausibility constraint,

making them a viable way of encoding images for neural applications [Van Rullen and Thorpe, 2001; Sen and Furber, 2009; Masmoudi et al., 2010]. Rank ~~order coding~~ Order Coding (ROC) can be performed using an algorithm known as the FoCal algorithm [Sen and Furber, 2009]. This algorithm models the foveola, the highest resolution area of the retina, with four ganglion cell layers each with a different scale of centre-surround receptive field [Kolb, 2003]. The detailed description of the algorithm was demonstrated by Garibaldi Pineda-García in the published paper [Liu et al., 2016] and the source Python scripts to transform images to ROC spike trains, and to convert the results into AER and PyNN spike source arrays, can be found in the dataset website.

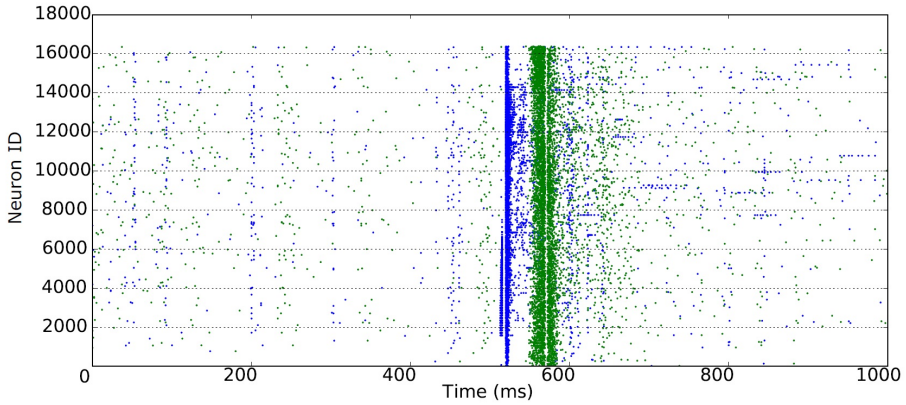
DVS Sensor Output with Flashing Input

The purpose of including the subset with DVS-recorded flashing digits is to promote research into rapid and accurate 'core recognition', thus to encourage research into non-rate-based algorithms to shorten the recognition time.

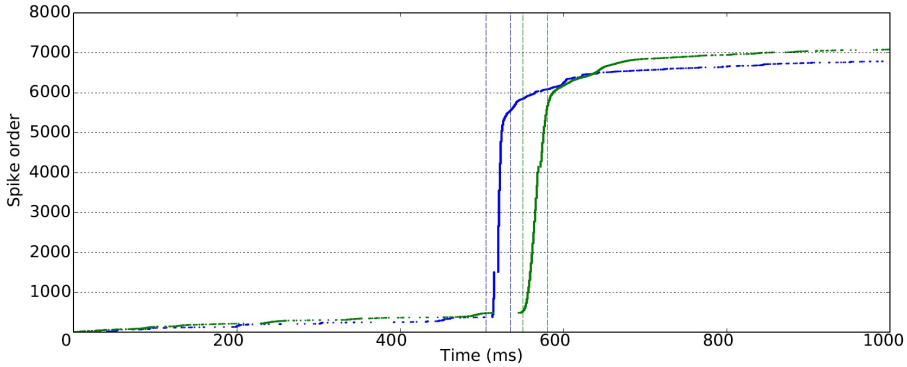
Each digit was shown alternating with a blank image and each display lasted one second. The digits were displayed on an LCD monitor in front of the DVS retina [Serrano-Gotarredona and Linares-Barranco, 2013] and were placed in the centre of the visual field of the camera. Since there are two spike polarities - 'ON' indicating an increase in the intensity while 'OFF' indicates a decrease - there are 'ON' and 'OFF' flashing recordings respectively per digit. In Figure 6.2, the burstiness of the spikes is illustrated where most of the spikes occur in a 30 ms time slot. In total, this subset of the database contains 2×60K recordings for training and 2×10K for testing.

DVS Sensor Output with Moving Input

The subset of DVS recorded moving digits is provided by Teresa Serrano-Gotarredona, and is presented to address the challenges of position- and scale- invariance in computer vision. MNIST digits were scaled to three different sizes, using smooth interpolation algorithms to increase their size from the original 28×28 pixels, and displayed on the monitor with slow motion. The same DVS [Serrano-Gotarredona and Linares-Barranco, 2013] used in Section 6.3.3 captured the movements of the digits and generated spike trains for each pixel in its 128×128 resolution. A total of 30K recordings were made: 10 digits, at 3 different scales, 1K different handwritten samples for each.



(a) The raster plot of the spike trains of a flashing digit.



(b) The same spike trains ordered by the spike sequence.

Figure 6.2: DVS sensor with flashing input. Blue is used for 'ON' events and green for 'OFF' events. (a) The raster plot shows spikes generated by individual neurons over time. It is hard to recognise the total number of spikes due to the large number of neurons involved in the figure. Thus all the spikes are ordered in time, and displayed in the figure below. (b) The raster plot shows the ordered spike sequence over time. The total number of spikes is around 7K for both 'ON' and 'OFF' events. The bursty nature of the resulting spikes is illustrated, where most of the spikes occur in a 30 ms time slot.

6.4 Performance Evaluation

New concerns about the latency and energy cost arise over performance assessment on SNNs, thereby highlighting the advantages of spike-based processing [Tan et al., 2015]. Therefore we propose corresponding evaluation metrics and suggest a sufficient description of SNN models in this section. Once a model is implemented on a neuromorphic platform, the hardware performance can also be evaluated by running the particular model. This model-specific assessment provides more robust comparisons between hardware platforms by using the same network topology, neuron and synaptic

models, and learning rules. Consequently, a complementary evaluation methodology is essential to provide common metrics and assess both the model-level and hardware-level performance.

6.4.1 Model-level Evaluation

A suggested description of an SNN model is shown in Table 6.1 where the performance evaluation metrics are in bold and the SNN specific description is in italics. Because SNNs introduce the time dimension and spike-based processing, additional performance metrics become relevant in addition to classification accuracy: recognition latency and the number of synaptic events.

Recognition latency measures how fast spikes are conveyed through the layers of network to trigger the recognition neurons. DiCarlo et al. [2012] considered the rapid (<200 ms) and accurate vision recognition in the brain as the essential problem of object recognition. For real-time systems with live visual inputs, such as robotic systems, a short response latency helps make fast decisions and take rapid action. The latency is measured as the time difference between the first spike generated by the output layer and the first spike from the input layer.

A spike event is a synaptic operation evoked when one action potential is transmitted through one synapse [Sharp et al., 2012]. Fewer spike events imply lower overall neural activity and lower energy consumption. The number of synaptic events can be measured as 'Sopbs', synaptic operations per biological second.

Alongside the SNN evaluation metrics, a sufficient description of a network model is required so that other researchers can reproduce it and compare it with other models. First of all, the input of an SNN model is specified. The description includes the transformation method for converting raw images to spike trains, and the preprocessing either to images or spikes. Filtering the raw image may ease the classification/recognition task while adding noise may require more robustness in the model. Secondly, as with the evaluation of conventional artificial neural networks, a description of the network characteristics provides the basis for the overall performance evaluation. Sharing the ~~designs~~ network details not only makes the model reproducible but also inspires fellow scientists to ~~bring new points of view to the problem,~~ build up new solutions based on existing work, thereby generating a positive feedback loop where everybody wins.

Table 6.1: SNN descriptions at the model level

Input	*converting methods* preprocessing
Network	topology *neuron and synaptic type*
Training	supervised or not *learning rule* *biological training time*
Recognition	**classification accuracy** ***response latency*** ***number of synaptic events*** *biological testing time* *input spiking rate*

The main characteristics include the network topology and the neural and synaptic models. The network topology defines the number of neurons used for each layer and the connections between layers and neurons. It is essential to state the types of neural and synaptic model (e.g. current-based LIF neuron) utilised in the network and the parameters configuring them, because neural activities differ significantly between configurations. Any non-neural classifier, sometimes added to aid the design or enhance the output of the network, must also be specified. Thirdly, the training procedure determines the recognition capability of a network model. Specifying the learning algorithm with its mechanism (supervised, semi-supervised and unsupervised) helps the reader understand the core features of the model. A detailed description of new spike-based learning rules will be a great contribution to the field due to the present paucity of spatio-temporal learning algorithms. Most publications reflect the use of adaptations to existing learning rules; details on these modifications should be clear and unambiguous. In conventional computer vision, the number of iterations of training images presented to the network play an important role. Similarly, the biological training time determines the amount of information provided for training an SNN. Finally in the testing phase, as well as the performance evaluation metrics stated above, specific configurations of the input spikes are also essential. This includes details of

the way samples are presented to the network: spiking rates, and biological time per test sample. The combination of these two factors determines how much information is presented to the network. Following the formatted evaluation as in Table 6.1, Table 6.2 lists a few SNN models for MNIST classification, although some details are missing.

6.4.2 Hardware-level Evaluation

A direct comparison between neuromorphic platforms is a non-trivial task due to the different hardware implementation technologies as mentioned in Section 2.3.2. Table 6.3 attempts to describe the neuromorphic hardware platforms with reference to different aspects of SNN simulation. The scalability of a hardware platform determines the network size limit of a neural application running on it. Considering the various neural and synaptic models, plasticity learning rules and lengths of axonal delays, a programmable platform offers flexibility to support diverse SNNs while a hard-wired system supporting only specific models is advantageous due to its energy-efficiency and simpler design and implementation. The classification accuracy of an SNN running on a hardware system can be different from the software simulation, since hardware implementations may impose limits on the precision used for the membrane potentials of neurons (for the digital platforms) and the synaptic weights. Simulation time is another important measure when running large-scale networks on hardware. Real-time implementation is an essential requirement for robotic systems because of the real-time input from the neuromorphic sensors. Running faster than real time is attractive for large and long simulations. It is interesting to compare the performance of each platform in terms of energy requirements, especially if the platform targets mobile applications and robotics. Some researchers have suggested the use of energy per synaptic event (J/SE) [Sharp et al., 2012; Stromatias et al., 2013] as an energy metric because the large fan in and out of a neuron means that synaptic processing tends to dominate the total energy dissipation during a simulation. Merolla et al. [2014] proposed the number of synaptic operations per second per Watt (Sops/W). These two measures are equivalent, since J/SE$\times$Sops/W = 1.

Table 6.2: Model-level comparison

	Input	Network	Training	Recognition
[Brader et al., 2007]	Poisson	Two layer, LIF neurons bistable synapse	Semi-supervised, STDP, calcium LTP/LTD	96.5%
[Diehl et al., 2015a]	Poisson	Two layers, LIF neurons, inhibitory feedback	Unsupervised, WTA, STDP, 200K s times 15 iterations	95%
[Neftci et al., 2013]	Thresholding, Poisson	Two layers, RBM, LIF neurons	Event-driven contrastive divergence (eCD), unsupervised	91.9% 1 s per test
[Neftci et al., 2016]	Thresholding, Poisson	Two layers, RBM, LIF neurons	Synaptic Sampling Machine + eCD, unsupervised	95.8% 250 ms per test
[Stromatias et al., 2015b]	Poisson	Four layers, RBM, LIF neurons	Off-line trained, unsupervised	94.94% 16 ms latency 1.44M Sopbs
[Diehl et al., 2015b]	Poisson	Six layers, ConvNet, IF neurons	Off-line trained with ReLU, weight normalisation	99.1%, 0.5 s per test
[Zhao et al., 2015]	Thresholding or DVS	Simple (Gabor), Complex (MAX) and Tempotron	Tempotron, supervised	91.3%(Thresholding) 11 s per test 88.1%(DVS), 2 s per test
Chpt4	Poisson	Six layers, ConvNet, LIF neurons	Off-line trained with ReLU, Noisy Softplus fine-tune	99.07%, 1 s per test 35.24 ms 53.4M Sopbs
Chpt5	Poisson	Two layers, Autoencoders (AE), LIF neurons	Event-driven, spike-based AE 18K s training, unsupervised	94.72%, 1 s per test 21.68 ms 5.26M Sopbs
Case Study	Poisson	Fully connected decision layer, LIF neurons	K-means clusters, Supervised STDP 18K s of training	92.99% 1 s per test 13.82 ms latency 4.17M Sopbs

Table 6.3: Hardware-level comparison

	System	Neuron Model	Synaptic Plasticity	Precision	Simulation Time	Energy Usage
SpiNNaker [Stromatias et al., 2013]	Digital, Scalable	Programmable Neuron and Synapse, Axonal delay	Programmable learning rule	11- to 14-bit synapses	Real-time Flexible time resolution	8 nJ/SE
TrueNorth [Merolla et al., 2014]	Digital, Scalable	Fixed models, Config params, Axonal delay	No plasticity	122 bits params & states, 4-bit/ 4 values synapses[1]	Real-time	26 pJ/SE
Neurogrid [Benjamin et al., 2014]	Mixed-mode, Scalable	Fixed models, Config params	Fixed rule	13-bit shared synapses	Real-time	941 pJ/SE
HI-CANN [Schemmel et al., 2010]	Mixed-mode, Scalable	Fixed models, Config params	Fixed rule	4-bit/ 16 values synapses	Faster than real-time[2]	7.41 nJ/SE (network only)
HiAER-IFAT [Yu et al., 2012]	Mixed-mode, Scalable	Fixed models, Config params	No plasticity	Analogue neuron/synapse	Real-time	22-pJ/SE [Park et al., 2014]

[a] We consider them 4-bit synapses because it is only possible to choose between 4 different signed integers and whether the synapse is active or not.
[b] A maximum speed-up of up to 10^5 times real time has been reported.

However, the typical reported simulation time and energy use for the various platforms is under different SNN models, making the comparisons problematic. Model-specific hardware metrics would provide robust comparisons between platforms and expose how different networks influence the metrics on particular hardware. The proposed evaluation metrics consist of the **feasibility**, **classification accuracy**, **simulation time** and **energy use**. A particular SNN model is feasible to run on a particular hardware platform only when the network size is under the platform's limit, the neural and synaptic models are supported, and the learning rule is implemented. CA also plays a role in hardware evaluation because of the precision limits that may be imposed by the platform. Due to the limited hardware resources, simulation time may accelerate or slow down according to the network topology and spike dynamics. Similarly, energy costs vary with different networks and neural and synaptic models.

6.5 Results

In this section, we present a recognition SNN model working on the Poissonian subset of the NE15-MNIST dataset. The network components, training and testing methods are described along the lines set out in Section 6.4.1. The recognition result is evaluated using the proposed metrics: classification accuracy, response latency and number of synaptic events. As tentative benchmarks the models are implemented on SpiNNaker to assess the hardware-level performance against software simulators. Presenting proper benchmarks for vision recognition systems is still under investigation; the case study only make a first attempt.

The case study is a simple two-layer network where the input neurons receive Poisson spike trains from the dataset and form a fully connected network with the decision neurons. There is at least one decision neuron per digit to classify a test input. The neuron with the highest output firing rate classifies a test image as the digit it represents. The model utilises LIF neurons, and the parameters are all biologically valid, see the listed values in Table 6.4. The LIF neuron model follows the membrane potential dynamics (Equation 4.1):

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = V_{rest} - V + R_m I(t) \quad , \tag{6.1}$$

where τ_m is the membrane time constant, V_{rest} is the resting potential, R_m is the membrane resistance and I is the synaptic input current. In PyNN, R_m is represented by $R_m = \tau_m / C_m$, where C_m is the membrane capacitance. A spike is generated when the membrane potential goes beyond the threshold, V_{thresh} and the membrane potential then resets to V_{reset}. In addition, a neuron cannot fire within the refractory period, τ_{refrac}, after generating a spike.

The connections between the input neurons and the decision neurons are plastic, so the connection weights can be modulated during training with a multiplicative STDP learning rule, refer to Section 2.2.4 for more detail. The model is described with PyNN and the code is published in the Github repository with the dataset. As a potential benchmark, this system is composed of simple neural models, trained with standard learning rules and written in a standard SNN description language. These characteristics allow the same network to be tested on various simulators, both software- and hardware-based.

Both training and testing use the Poissonian subset of the NE15-MNIST dataset. This makes performance evaluation on different simulators possible with the unified spike source array provided by the dataset. In terms of this case study, the performance of the model was evaluated with both software simulation (on NEST [Gewaltig and Diesmann, 2007]) and on a hardware implementation (on SpiNNaker).

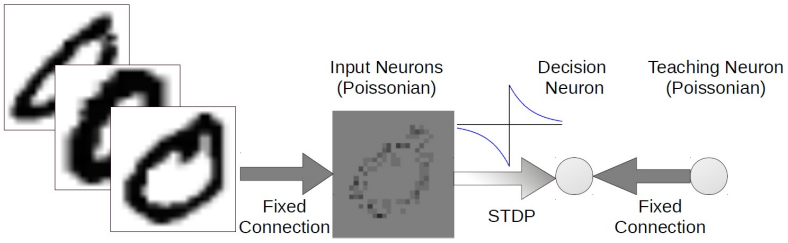
In order to fully assess the performance, different settings were configured on the network, such as network size, input rate and test image duration. For simplicity of describing the system, a set of standard configuration is used as the example in the following sections.

Table 6.4: Parameter setting for the current-based LIF neurons using PyNN.

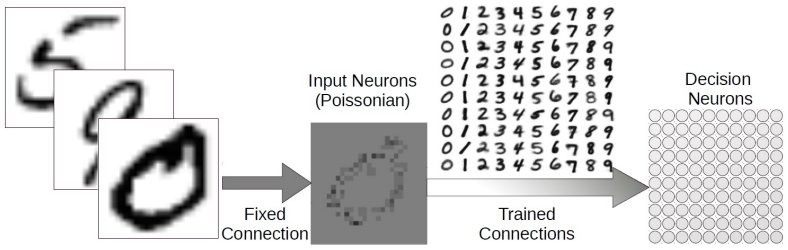
Parameters	Values	Description
C_m	0.25 nF	membrane capacitance
τ_m	20.0 ms	membrane time constant
τ_{refrac}	2.0 ms	refractory period
V_{reset}	-70.0 mV	resting membrane potential
V_{rest}	-65.0 mV	resetting membrane potential
V_{thresh}	-50.0 mV	membrane threshold
I_{offset}	0.0 nA	offset of current influx

6.5.1 Training

There are two layers in the model: 28×28 input neurons fully connect to 100 decision neurons. Each decision neuron responds to a certain digit template. In the standard configuration, there are 10 decision neurons responding to each digit with slightly different templates. Those templates are embedded in the connection weights between the two layers. Figure 6.3(a) shows how the connections to a single decision neuron are tuned.



(a) Training model of a single decision neuron.



(b) Testing model of the spiking neural network.

Figure 6.3: The (a) training and (b) testing model of the proposed case study SNN: STDP learning with supervised teaching signal.

The training set of 60K hand written digits are firstly classified into 100 classes, 10 subclasses per digit, using K-means clusters. K-means clustering separates a set of data points into K subsets (clusters) according to the Euclidean distance between them. Therefore each cluster tends to form a boundary within which the data points are near to each other. In this case, all the images of the same digit (a class) are divided into 10 subclasses by assigning K=10. Then the images in a certain subclass are used to train a template embedded in the synaptic weights to the corresponding decision neuron. The firing rates of the input neurons are assigned linearly according to their intensities and the total firing rate of all the 28×28 input neurons is normalised to 2K Hz, that is, the sum of the firing rates of all of the input neurons is 2K Hz. All the images together are presented for 18K s (about 300 ms per image) during training

and at the same time a teaching signal of 50 Hz is conveyed to the decision neuron to trigger STDP learning. The trained weights are plotted in accordance with the positions of the decision neurons in Figure 6.3(b).

6.5.2 Testing

After training the weights of the plastic synapses are set to static, keeping the state of the weights at the last moment of training. However, during training the synaptic plasticity holds a hard limit of 0 on the weight strength, thus excitatory synapses cannot change into inhibitory. To investigate how inhibitory connections influence the classification performance, the weak weights were set to negative with identical strengths. Results show that inhibitory synapses significantly reduced the output firing rates while keeping a good classification ability. Thus the strategy of replacing weak weights ~~to~~ with the same negative values was used throughout the case study.

The feed-forward testing network is shown in Figure 6.3(b) where Poisson spike trains are generated the same way as in the training with a total firing rate of 2K Hz per image. The input neurons convey the same spike trains to every decision neuron through its corresponding trained synaptic weights. One test trial contains 10K images in total and each image is presented once and lasts 1 s with a 0.2 s blank period between consecutive images. The output neuron with the highest firing rate determines which digit is recognised. With the standard training configuration, we compared the CA of different simulations of the same SNN model. Using the trained weights from the NEST simulation, the accuracy of the recognition on NEST reached 90.03%, and this accuracy was also achieved on SpiNNaker. When the network was both trained and tested on SpiNNaker the recognition accuracy was 87.41%. Using these weights in NEST yielded a similar result (87.25%). The reduction in CA using the SpiNNaker trained weights was due to precision loss caused by the limited fast memory and the necessity for fixed-point arithmetic to ensure real-time operation. It is inevitable that numerical precision will be below IEEE double precision at various points in the processing chain from synaptic input to membrane potential. The main bottleneck is currently in the ring buffer where the total precision for accumulated spike inputs is 16-bit, meaning that individual spikes are realistically going to be limited to 11- to 14-bit depending upon the probabilistic headroom calculated as necessary from the

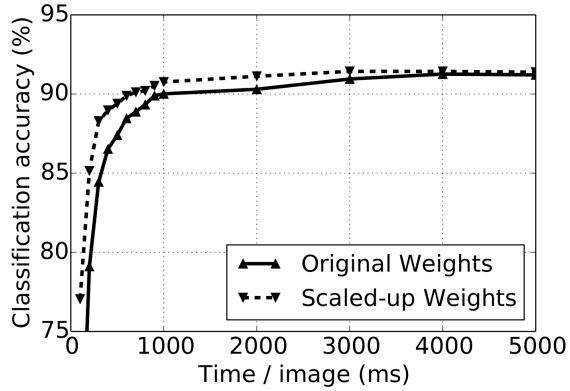
network configuration and spike throughput [Hopkins and Furber, 2015].

6.5.3 Evaluation

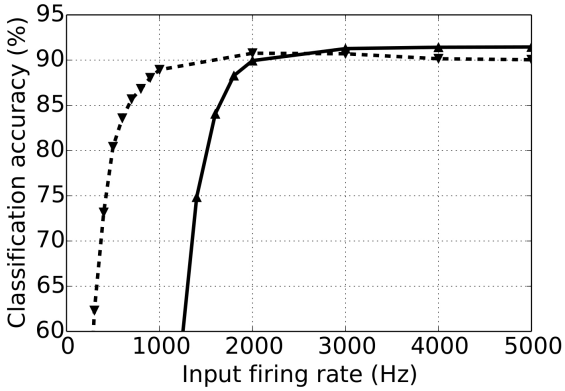
Evaluation starts from the model-level, focusing on the spike-based recognition analysis. As mentioned in Section 6.4.1, CA, response time (latency) and the total number of synaptic events are the main concerns when assessing the recognition performance. In our experiment, two sets of weights were applied: the original STDP trained weights, and the ~~scaled-up weights which are~~ same weights strengthened 10 times ~~stronger~~(which we call scaled-up weights). The spike rates of the test samples were also modified, ranging from 10 to 5K Hz.

We found that accuracy depends largely on the time each sample is exposed to the network and the sample spike rate (Figure 6.4). Figure 6.4(a) shows that the CA is better as exposure time increases. The longer an image is presented, the more information is gathered by the network, so the accuracy climbs. Classification accuracy also increases when input spike rates are augmented (Figure 6.4(b)). Given that the spike trains injected into the network are more intense, the decision neurons become more active, and so does the output disparity between them. Nonetheless, it is important to know that these increases in CA have a limit, as is shown in the aforementioned figures. With stronger weights, the accuracy is much higher when the input firing rate is less than 2K Hz.

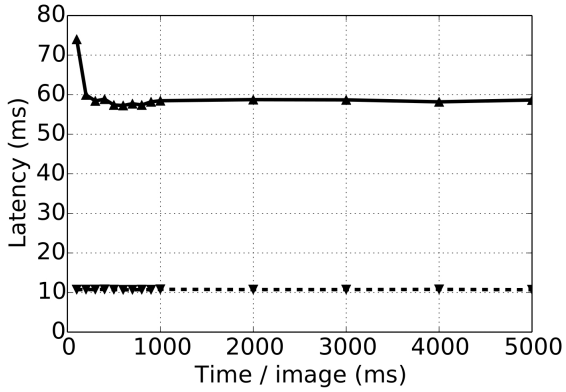
The latency of an SNN model is the result of the input firing rates and the synaptic weights. We measured the latency of each test by getting the time difference of the first spike generated by any decision neuron in the output layer and the first spike of the input layer. As the input firing rates grow, there are more spikes arriving at the decision neurons, triggering them to spike sooner. A similar idea applies to the influence of synaptic weights. If stronger weights are taken, then the membrane potential of a neuron reaches its threshold earlier. Figure 6.4(d) indicates that the latency is shortened with increasing input firing rates with both the original and scaled-up weights. When the spiking rate is less than 2K Hz, the network with stronger weights has a much shorter latency. As long as there are enough spikes to trigger the decision neurons to spike, increasing the test time will not make the network respond sooner (Figure 6.4(c)).



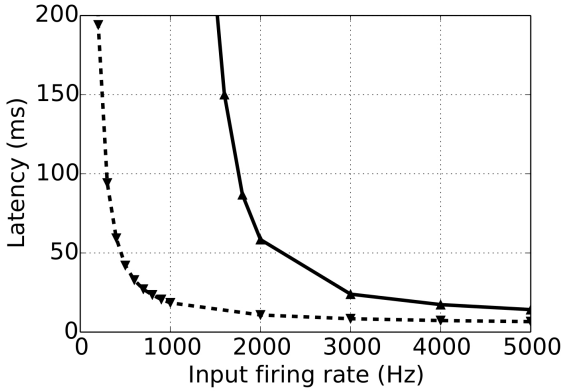
(a) Accuracy changes against test time.



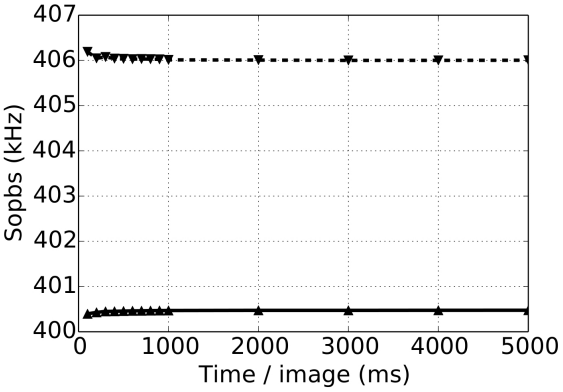
(b) Accuracy changes against firing rate.



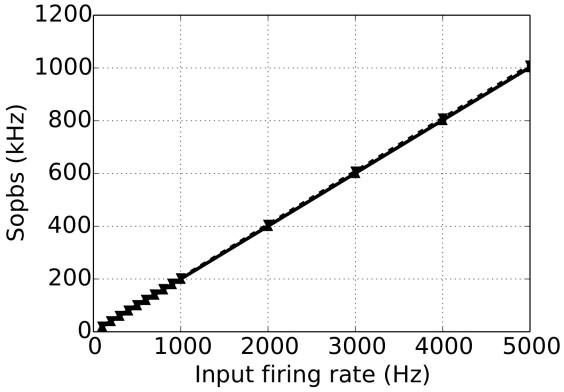
(c) Latency stablises against test time.



(d) Latency changes against firing rate.



(e) Event rate stablises against test time.



(f) Event rate changes against firing rate.

Figure 6.4: Accuracy, response time (latency) and synaptic event rate (Sopbs) change over test time and input firing rate per test image. The test time is the duration of the presence of a single test image, and the input firing rate is the summation of all the input neurons. Original trained weights are used (up-pointing triangles with solid line) as well as the scaled up ($\times 10$) weights (down-pointing triangles with dashed line).

At the default configuration of the SNN model, each input neuron connects to all of the 100 decision (output) neurons with both excitatory and inhibitory projections.

Thus the synaptic events happening in the inter-layer connections are 200 (100×2) times the total input firing rate. Figure 6.4(e) shows the stable Sopbs of the entire network when the input firing rate is held at 2K Hz and the test time increases. The firing rates of the output layer are relatively small, and are 0.1% and 1.5% of the total Sopbs using original and scaled-up weights respectively. The variations in the total Sopbs lie in the firing rate of the output layers only, and the stronger connections lead to the higher firing rates. Likewise, the output neurons are more active with stronger connection weights, and the gap widens as the input firing rate increases, see Figure 6.4(f). Although the variations in the Sopbs climb to around 8 kHz, it is not obvious in the figure because the output firing rates are relatively low and therefore so are the differences.

The network size not only influences the accuracy of a model but also the time taken for simulation on specific platforms, thus impacting the energy usage on the hardware. For the purpose of comparing the accuracy, simulation time, number of synaptic events and energy usage, different configurations have been tested on NEST (working on a PC with CPU: i5-4570 and 8G bytes memory) and on SpiNNaker. The same experiment was run 4 times with different random seeds; the average performance estimation is listed in Table 6.5. The input rates in all of the tests are 5K Hz, and each image is presented for 1 s with a 0.2 s blank period between consecutive images during which the model receives no input. The configurations only differ in the number of templates (subclasses/clusters) per digit.

As the network size grows there are more decision neurons and synapses connecting to them, thus the simulation time on NEST increases. On the other hand, SpiNNaker works in (biologically) real time and the simulation time becomes shorter than the NEST simulation when 1K patterns per digit (1K decision neurons per digit) are used. The NEST simulation was run on a desktop PC, and the power use was measured by a meter socket and estimated by subtracting the usage of idle OS operation from the usage running the simulation. In doing so, the power consumption of the resources needed to run the simulation is better approximated. The SpiNNaker test was run on a Spin4 board which has 48 chips and exposed pins to measure electrical quantities. A built-in Arduino board provided a measurement read out of the power usage of the chips. For the same goal of estimating just the required resources, only the active

Table 6.5: Comparisons of NEST (N) on a PC and SpiNNaker (S) performance averaged over 10 trials.

Subclasses per digit		1	10	50	100	1000
Avg. response latency (ms)		18.03	14.25	13.82	13.57	13.15
Avg. synaptic events (Sopbs)		83,691.48	835,274.98	4,173,392.03	8,343,559.69	83,385,785.67
Accuracy (%)	N	79.63±0.23	91.42±0.13	92.99±0.15	87.05±0.21	89.63±0.08
	S	79.57±0.31	91.39±0.09	92.99±0.08	87.00±0.26	89.58±0.24
Avg. SIM time (s)	N	445.09	503.21	767.67	1,131.09	12,027.75
	S			12K		
Power (W)	N	20	20	20	19	17
	S	0.38	0.38	0.41	0.44	1.50
Energy (KJ)	N	8.90	10.06	15.34	21.50	208.25
	S	4.56	4.56	4.92	5.28	18.00

chips were measured. Even with the smallest network, SpiNNaker wins in the energy cost comparison, see Figure 6.5. Among different network configurations, the model with 500 decision neurons (50 clusters per digit) reaches the highest recognition rate of 92.99% on average having a latency of 13.82 ms mean and 2.96 ms standard deviation. And there are standard deviations of 2.57% on CA and of 1.17 ms on the latency over 10 testing digits. The total number of synaptic events is around 4.17M Sopbs, where only 7K spikes are generated in the output layer. The NEST simulation costs 767.67 s on average for the entire 12K s biological-time test, 20 W in power use on the PC and 15.35 KJ of energy, while SpiNNaker works in real time using 4.92 KJ of energy at a power of 0.41 W (see Table 6.5). This result provides a baseline for comparison with other SNN models and neuromorphic hardware, and no optimisation is applied.

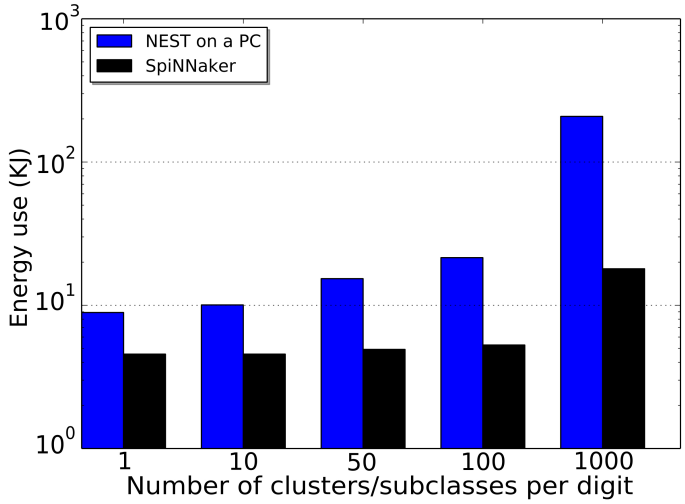


Figure 6.5: Energy usages of different network size both using NEST (blue) on a PC and SpiNNaker (black).

6.6 Summary

This chapter put forward the NE dataset as a unified resource to quantitatively measure progress within the field of neuromorphic vision. It enables not only objective comparisons among SNNs, but also between conventional ANNs and these spike-based models. In addition, as an evolving dataset, it catches up with the cutting-edge technologies thereby promoting future research in NE.

The current dataset NE-15 contains four spike-based subsets aiming at different purposes: (1) the Poissonian subset is intended for testing rate-based recognition

methods; (2) the rank-order coded subset aims at spatio-temporal algorithms; (3) the DVS recorded flashing input subset encourages fast recognition; and (4) the other DVS recorded moving input is designed for mobile neuromorphic robots and is a suitable test for invariant object recognition.

The proposed complementary evaluation methodology is the first to analytically assess SNN models and the neuromorphic hardware simulators. We carefully selected response latency and the number of synaptic events for model-level evaluation; plus, simulation time and energy usage for evaluating hardware performance. These special evaluation metrics highlight the strengths of spike-based AI tasks.

A potential benchmark system is evaluated using the Poissonian subset of the NE15-MNIST dataset. This example demonstrates a recommended way of using the dataset, describing the SNN models and evaluating the system performance. The case study provides a baseline for robust comparisons for SNN models and their hardware implementations, and also validates the feasibility of the database and its evaluation methodology.

Building up this dataset is a team work, I claim my own contributions are two sunsets of the dataset out of four, the complementary evaluation methodology and the on-line learning case study.

Chapter 7

Conclusion and Future Work

In this chapter, we will confirm the hypotheses proposed in Chapter 1 to answer the thesis question: how to close the gap between the cognitive capabilities of Spiking Neural Networks (SNNs) and Artificial Neural Networks (ANNs) on Artificial Intelligence (AI) tasks. This involves introducing sub-topics on how we arrived at this confirmation, what are the main contributions, and how this work challenges previous research. This will be followed by a statement of the current limitations of our study, potential methods for tackling these limitations, and directions for further research.

7.1 Confirming Research Hypotheses

1. Deep SNNs can be successfully and simply trained off-line where the training takes place on equivalent ANN and the tuned weights then transferred back to the SNNs, thus making them as competent as ANNs in cognitive tasks.

The key problem of such an off-line method lies in the transformation of ANN models to SNNs. In Chapter 4, we broke down the problem into two smaller parts and solved them with proposed novel activation functions: Noisy Softplus (NSP) and the Parametric Activation Function (PAF). NSP accurately models the output firing activity in response to the current influx of a Leaky Integrate-and-Fire (LIF) neuron. It tackles the first problem of modelling discrete, spike-based neural computation with continuous activation functions of abstract values in ANNs. Next, PAF addresses the other problem of mapping these numerical values to concrete physical units in SNNs:

input currents in nA and output firing rates in Hz. Therefore, a standard LIF neuron of biologically-valid parameters can be represented as a PAF-NSP neuron in ANN. Consequently, an ANN comprised of PAF-NSP neurons can work equivalently to an SNN of LIF neurons; and the weights of this ANN model could be transferred to the SNN without any transformation. Moreover, the training of PAF-NSP neurons can be generalised to a PAF version of conventional activation functions, which greatly reduces the computational complexity.

On one hand, this research contributes to the NE community a simple off-line SNN training method. It enables any feed-forward SNN to be modelled and trained on an equivalent ANN, and thus resolves the difficulties of transforming ANN models to SNNs. On the other hand, the method is generalised in terms of spiking neural models and hardware platforms, since it works on standard LIF neurons which are supported by most neuromorphic hardware. Hence, AI engineers are capable of implementing their ANN models on NE platforms without knowledge of SNN or hardware-specific programming, thereby benefiting from neuromorphic hardware in many ways: such as energy-efficiency, biological realism, low latency and real-time processing.

Among the existing approaches, this method is the first and only one that unifies the representation of neurons in ANNs and the spiking ones in SNNs using abstract activation functions, while some other approaches [Jug et al., 2012; Hunsberger and Eliasmith, 2015] directly handle physical units. In terms of modelling accuracy, NSP not only takes the noise of the current influx brought by the random arrival of spikes into account, which is missing in soft LIF [Hunsberger and Eliasmith, 2015], but also includes coloured noise where the Siegert formula [Jug et al., 2012] only works on white noise. Regarding simplicity, this training method performs effectively using Rectified Linear Units (ReLUs), however, other studies [Jug et al., 2012; Hunsberger and Eliasmith, 2015] suffer from high computational complexity. In addition, thanks to the accurate modelling of PAF, the trained weights are ready to use on SNNs, while previous approaches [Cao et al., 2015; Diehl et al., 2015b] require an extra step to adapt the trained weights to the SNN. Moreover, comparing to the requirements of specific neural models [Cao et al., 2015; Diehl et al., 2015b] and the hardware-specific training methods [Diehl et al., 2016b,a; Esser et al., 2015], our approach is generalised to target standard LIF neurons and neuromorphic hardware platforms.

To validate the cognitive capability of the SNN models, we compared the classification accuracy of the spiking ConvNets to the non-spiking ANNs. The performance was nearly equivalent, and the best classification accuracy achieved 99.07% on the MNIST task which outperformed state-of-the-art SNN model of LIF neurons [Hunsberger and Eliasmith, 2015] and equalled the best result using simplified IF neurons [Diehl et al., 2015b]. Therefore, we confirm the hypothesis with the first contribution of this thesis: providing the NE community a simple, but effective, generalised, off-line deep SNN training method.

2. Unsupervised Deep Learning modules can be trained on-line on SNNs with biologically-plausible synaptic plasticity to demonstrate a learning capability equivalent to ANNs.

On-line training aims to bring biologically-plausible learning rules to SNNs to equip neuromorphic computers with genuine learning capabilities. Instead of transforming off-line trained ANN models to SNNs, on-line methods face the problem of translating numerical computations for training Deep Learning modules into spike-based synaptic plasticity rules. Multiplication is the core operation in the unsupervised Deep Learning algorithms, Autoencoders (AEs) and Restricted Boltzmann Machines (RBMs)~~: $\Delta w \propto ab - cd.$~~. In Chapter 5 we found that the product of two numerical values could be represented with rate multiplication of a pair of rate-coded spike trains; and the proposed formalised Spike-based Rate Multiplication (SRM) method precisely transformed the product of rates to the number of simultaneous spikes generated from a pair of connected spiking neurons. Most importantly, the simultaneous events were captured by the weight change of the synaptic connection using the Spike-Timing-Dependent Plasticity (STDP) learning rule. Therefore, the SRM tackles the problem of translating the weight tuning from numerical computations to event-based, biologically-plausible learning rules in SNNs.

Spiking AEs and RBMs can be trained with SRM by sharing the synaptic weights between ab and cd, and applying symmetric learning rates $\pm\eta_s$ on the weight changes respectively. The performance approaches the same, sometimes even superior, classification and reconstruction capabilities compared to their equivalent non-spiking models. In addition, the numerical analysis of the proposed algorithm accurately estimates

the learning rate $\pm\eta_s$, thus closely mimicking the learning behaviour of the AE and RBM modules, and improves the learning performance compared to existing methods.

Thanks to the accurate parameter estimation, the classification results (94.72% for SAE and 94.35% for SRBM) have outperformed most existing on-line models. Neil [2013] ignored the ~~mathematical analysis~~ model formalisation and accurate parameter settings thus achieved its best performance only at 81.5%. Neftci et al. [2013] conducted training on a recurrent network which was more biologically plausible, but with a different goal of neural sampling this method achieved a worse classification at 91.9%. Their extended work [Neftci et al., 2016] improved the performance to the state-of-the-art, 95.6%, but requires extra Deep Learning technique, dropout [Srivastava et al., 2014], and more training epochs. Nevertheless, Neftci et al. [2016] did not compare the trained SNN to the non-spiking counterpart, RBM-dropout; while our method closely reproduced the learning curves of the equivalent ANNs. Moreover, our proposed method successfully decorrelates spike trains, thereby solving the problem of performance drop caused by the spike correlation. Therefore, this method enables the genuine 'live and learn' in on-line SNN training. Furthermore, in theory our method works on any spiking neuron model and requires a simple rectangular STDP, thus can be easily implemented on general neuromorphic hardware. However, it is more difficult for other existing approaches [Neftci et al., 2013, 2016] to work on hardware platforms since they ask for either specific neural/synaptic models or extra external signals to control the learning.

Our second contribution is the formalisation of an STDP-based unsupervised learning algorithm for spiking AE (SAE) and spiking RBM (SRBM). The promising results of equivalent or even superior classification and reconstruction capabilities of SAEs and SRBMs compared to their conventional ANN-based methods confirms the hypothesis that SNNs have learning ability as competent as deep ANNs.

3. A new set of spike-based vision datasets can provide resources and corresponding evaluation methodology to support objective comparisons and measure progress within the rapidly advancing field of NE.

In Chapter 6 we presented a dataset containing four different spike representations of the MNIST stationary hand-written digit database. The Poissonian subset is intended

for benchmarking existing rate-based recognition methods. The rank-order coded subset, FoCal, encourages research into spatio-temporal algorithms for recognition applications using only small numbers of input spikes. Fast recognition can be verified on the DVS recorded flashing input subset, since just 30 ms of useful spike trains are recorded for each image. Another DVS recorded moving input can be used to test mobile neuromorphic robots. Orchard et al. [2015] presented a neuromorphic dataset using a similar approach to capture moving input, but the spike trains were obtained with micro-saccades. This dataset aims to convert static images into neuromorphic vision input, while the recordings of moving input in ~~our paper~~ this thesis are intended to promote position-invariant recognition. Therefore, the datasets complement each other.

The proposed complementary evaluation methodology is essential to assess both the model-level and hardware-level performance of SNNs. In addition to classification accuracy, response latency and the number of synaptic events are specific evaluation metrics for model-level spike-based processing; plus, simulation time and energy usage are for evaluating hardware performance. These carefully selected evaluation metrics highlight the strengths of spike-based AI tasks. It is also important to describe an SNN model in sufficient detail to share the network design. The network size of an SNN model that can be built on a hardware platform will be constrained by the scalability of the hardware. Neural and synaptic models are limited to those that are physically implemented, unless the hardware platform supports programmability. Any attempt to implement an on-line learning algorithm on neuromorphic hardware must be backed by synaptic plasticity support. Therefore running an identical SNN model on different neuromorphic hardware exposes the capabilities of such platforms. If the model runs smoothly on a hardware platform, it then can be used to benchmark the performance of the hardware simulator in terms of simulation time and energy usage. Classification accuracy is also a useful metric for hardware evaluation because of the limited precision of the membrane potential and synaptic weights.

A third contribution of the thesis provides the NE community with a dataset and its corresponding evaluation methodology for comparisons of SNN models and NE platforms, thus to measure the progress towards Neuromorphic Cognition. The successful baseline test of a benchmark system has been evaluated using the Poissonian subset

of the NE15-MNIST dataset, which validates the feasibility of the database and its evaluation. There, we confirm the hypothesis that the dataset provides resources and supports fair comparisons among SNN models and their hardware implementations.

7.2 Future Work

Though the research aim has been mostly achieved by the work presented in this thesis, some limitations still remain to be addressed in the future. In addition, this work has inspired new directions for future work to continue and expand the current research.

7.2.1 Off-line SNN Training

The current limitation prohibiting the off-line SNN training method from wide use lies in the lack of supporting tools. This requires the development of a set of software and libraries. Besides, we will look into networks with feedback connections, Recurrent Neural Networks (RNNs), thus to adapt the off-line training method to most network architectures. Moreover, some initial studies following our research have successfully applied the proposed method to various applications.

Supporting software tools. The plan of developing software and libraries in the near future includes:

- parameter calibration of p for PAF given LIF neuron configurations, which involves automatic SNN simulations with different levels of current and noise, followed by curve fit with the activation function NSP.
- supporting libraries for SNN training in popular Deep Learning platforms, e.g. Tensorflow [Abadi et al., 2016], which will include the proposed activation functions NSP and PAF in the ANN models.
- a unified template to describe any ANN model and an automation tool that reads platform-dependent trained models into the designed template. The tool will not only help to translate ANN models to SNNs, but also contribute to the cross-platform transfer learning and use of pre-trained models in Deep Learning.

- a translation tool that converts the tuned ANN models into SNNs described in the PyNN [Davison et al., 2008] language, thus the SNN model can run on any software simulator or neuromorphic hardware as long as PyNN is supported.

Recurrent Neural Networks. It is difficult to make SNNs work with recurrent architectures, since a spiking neuron simultaneously takes input from both the lower level on the feed-forward path and the upper level on the feedback path. However, in ANNs the feed-forward and feedback paths work alternately on separate steps. Therefore, the proposed SNN training method only applies to feed-forward networks. One of the major future goals is to propose a general method to run recurrent SNNs which perform equivalently to RNN models.

Applications. The simple off-line SNN training method has enabled and encouraged interesting applications to run in SNNs.

- Ensemble models [Krogh and Vedelsby, 1995] have been trained with NSP neurons and will be transferred onto the SpiNNaker system [Furber et al., 2014]~~to~~. Usually an ensemble model runs several identical neural networks with statistical features in parallel, thus occupies huge computing resources. Operating ensemble models on SNNs can take the advantages of energy-efficiency and massively-parallel processing on Neuromorphic Hardware.

- Speech recognition of simulated spikes from a cochlea model has achieved a promising accuracy at the initial test-idea stage. ~~The~~ We used spike count per frame per frequency channel as a pixel in a spectrum image, and trained CNNs based on these spectrum images. The trained weights were transferred directly to an equivalent convolutional SNN. The next step is to implement the model entirely on neuromorphic hardware including the cochlea and the SNN, thus to run it in real time.

- An 18-layer residual network [He et al., 2016] has been trained for the task of recognising human facial expressions using the KDEF dataset [Lundqvist et al., 1998]. Notably, this is, so far, the deepest network trained with NSP and the recognition performance (94.95%) has outperformed ReLU (92.45%). It will

be interesting to analyse the differences of recognition accuracy and robustness using NSP and other activation functions. Thus it may answer the question how the brain delivers strong cognitive ability with stochastic and noisy signals.

- A further goal is to implement deep SNNs fit for ImageNet [Deng et al., 2009] tasks, which will also require modelling various functions of Deep Learning on spiking neurons, such as max-pooling and batch normalisation.

7.2.2 On-line Biologically-plausible Learning

The proposed on-line learning method is limited to rate coding and greedy layer-wise training. Thus, we will explore alternative coding schemes and training algorithms to increase the effectiveness of the on-line learning methods. In addition, we will merge novel Deep Learning techniques into the biologically-plausible SNN training to improve the cognitive performance and also decrease the energy consumption when running on neuromorphic hardware.

Beyond rate coding. Although rate coding has shown good performance on training spiking Deep Learning modules on-line, time-based coding and rank-order coding which carry more information per spike, are expected to have better or faster learning capabilities [Gautrais and Thorpe, 1998]. We have proposed a similar on-line learning algorithm for training precise-timing based spiking AEs and RBMs, which although still in the test-idea stage, result of the prototype has shown much faster learning speed than the rate-coding mechanism.

Backpropagation alternatives. The STDP rule usually works locally with a teaching signal in supervised learning, however, error backpropagation (BP) does not provide the teaching targets for all the hidden units of a network. Thus, BP with gradient descent is believed to be difficult to transfer to SNNs and alternative methods with local training algorithms are preferred. Despite the success of greedy layer-wise training of AEs and RBMs, Random Back-Propagation (RBP) is also an alternative to backward targets with fixed random weights for hidden layers. Therefore, RBP fits to the local learning rules of synaptic plasticity in SNNs. Initial work [Samadi et al., 2017; Neftci et al., 2017] has shown that these random feedback weights work effectively to

replace precise BP, but endures complex neuron/synaptic models and network architectures. In the future, we will continue the investigation in BP alternatives within on-line SNN training.

Biologically-plausible Reinforcement Learning. The modulation of STDP by a third factor such as dopamine has potentially interesting functional consequences that turn STDP from unsupervised learning into a reward-based learning paradigm [Izhikevich, 2007] which addresses Reinforcement Learning. Merging advanced neuroscience findings and Deep Learning mechanisms onto on-line SNN training will be the trend for future work.

State-of-the-art Performance. Synaptic Sampling Machines (S2Ms) [Neftci et al., 2016] employing a dropout [Srivastava et al., 2014] mechanism which hugely improved the performance on MNIST tasks from 91.9% to 95.6%. Thus applying novel Deep Learning techniques for SNN training is also in the future work to improve the state-of-the-art performance..

Genuine Learning on Neuromorphic Hardware. On-line training is a necessary means towards Neuromorphic Cognition where the hardware computers learn while they operate. The main concerns are the learning speed and the power consumption, which will be compared to off-line training methods and conventional ANN training. It is still an open question how to perform the human-level cognition and at the same time achieve the low power cost by on-line learning.

7.2.3 Evaluation on Neuromorphic Vision

Since new problems will continue to arise before vision becomes a solved problem, the NE dataset will evolve as research progresses. The number of vision datasets will increase and the corresponding evaluation methodologies will evolve. The conversion methods for transforming images and videos into spike trains will advance.

Face recognition dataset. As mentioned in Section 6.1, face recognition has become a hot topic in SNN approaches, however there is no unified spike-based dataset to benchmark these networks. Thus, the next step is to include face recognition

databases. While viewing an image, saccades direct high-acuity visual analysis to a particular object or a region of interest and useful information is gathered during the fixation of several saccades in a second. It is possible to measure the scan path or trajectory of the eyeball and those trajectories show particular interest in eyes, nose and mouth while viewing a human face [Yarbus, 1967]. Therefore, our plan is also to embed modulated trajectory information to direct the recording using DVS sensors to simulate human saccades.

Converting images to spikes. Although Poisson spikes are the most commonly used external input to an SNN system, there are several *in-vivo* recordings in different cortical areas showing that the inter-spike intervals (ISI) are not Poissonian. Thus Deger et al. [2012] proposed new algorithms to generate superposition spike trains of Poisson Processes with Dead-time (PPD) and Gamma processes. Including novel spike generation algorithms in the dataset is one aspect of future work.

Invariant object recognition Each encounter of an object on the retina is unique, because of the illumination (lighting condition), position (projection location on the retina), scale (distance and size), pose (viewing angle), and clutter (visual context) variabilities. The brain, however, recognises a huge number of objects rapidly and effortlessly even in cluttered and natural scenes. To explore invariant object recognition, the dataset will include the NORB (NYU Object Recognition Benchmark) dataset [LeCun et al., 2004], which contains images of objects that are first photographed in ideal conditions and then moved and placed in front of natural scene images.

Video processing Action recognition will be the first problem of video processing to be introduced in the dataset. The initial plan is to use the DVS retina to convert the KTH [Schüldt et al., 2004] and Weizmann [Blank et al., 2005] benchmarks to spiking versions. Meanwhile, using the proposed software DVS retina [Garibaldi et al., 2016] to transform frames into spike trains is also on the schedule. By doing this, a huge number of videos, such as those in YouTube, can be converted automatically into spikes, therefore providing researchers with more time to work on their own applications.

Sharing and collaboration Overall, it is impossible for the dataset proposers to provide enough datasets, converting methods and benchmarking results, thus we encourage other researchers to contribute to the dataset allowing future comparisons using the same data source. They can also share their spike conversion algorithms by generating datasets to promote the corresponding recognition methods. Neuromorphic hardware owners are welcome to provide benchmarking results to compare their system's performance.

7.3 Closing Remarks

Energy-efficient neuromorphic hardware platforms have been successfully used to support large-scale simulations of biologically-plausible SNNs for brain understanding, but they are still far from intelligent enough to achieve Neuromorphic Cognition. Meanwhile, Deep Learning techniques in the field of ANN have driven simple rate-based artificial neurons to surpass human-level capabilities in cognitive tasks, e.g. vision. Thus, to equip these powerful brain-inspired neuromorphic computers with cognitive capabilities, this thesis contributes to close the gap of the performance between SNNs and ANNs on AI tasks.

One of the major contributions is a simple off-line SNN training method which models and trains any feed-forward SNN on an equivalent ANN, and enables the trained weights work equivalently on the SNN without any conversion. It significantly simplifies the development of AI applications on neuromorphic hardware thanks to the simple training process and the use of standard LIF neurons which are supported by most neuromorphic hardware platforms. The success of the generalised off-line method paves the way to energy-efficient AI from mobile devices to huge computer clusters.

Taking one step towards Neuromorphic Cognition, the on-line learning method has also been investigated to train Deep Learning modules on SNNs. The proposed method tackles the problem of accurately translating the weight tuning from numerical computations to event-based, biologically-plausible STDP rules in SNNs; and is the first to address the continuous performance drop caused by the spike correlations. In doing so, it directs the future work to formalise event-based learning to closely mimic the numerical computations in conventional ANNs. The development of on-line

training will equip the neuromorphic computers with genuine learning capabilities.

Last but not least, this work contributes to the NE community with a ~~uniform~~ unified dataset to evaluate SNNs' performance at both model and hardware level. The corresponding evaluation methodology will promote meaningful comparisons between these proposed SNN models and other existing methods within this rapidly advancing field. Moreover, we hope to provide objective comparisons between conventional ANNs and SNNs, in order to give prominence to low latency and energy consumption on neuromorphic hardware.

Appendix A

Detailed Derivation Process of Equations

In Chapter 3, we investigated the derivative of the log-likelihood with the respect to a parameter θ in general format (Equation 3.18) and in RBM (Equation 3.19). The following Equations A.1 and A.2 give the detailed derivation process.

In energy-based models, the probability of data point x is defined by a model function $f(x)$, its energy function $E(x)$ and a partition function Z which normalises the model function to possibilities by adding up all possible $f(x)$. The second term of the right hand side in Equation 3.18, the derivative of the log partition function Z, is as follows:

$$\begin{aligned}
\frac{\partial \log Z(\Theta)}{\partial \theta} =& \frac{1}{Z(\Theta)} \frac{\partial Z(\Theta)}{\partial \theta} \\
=& \frac{1}{Z(\Theta)} \sum_{\mathbf{x}} \frac{\partial f(\mathbf{x} \mid \Theta)}{\partial \theta} \\
=& \frac{1}{Z(\Theta)} \sum_{\mathbf{x}} f(\mathbf{x} \mid \Theta) \frac{1}{f(\mathbf{x} \mid \Theta)} \frac{\partial f(\mathbf{x} \mid \Theta)}{\partial \theta} \\
=& \sum_{\mathbf{x}} \frac{f(\mathbf{x} \mid \Theta)}{Z(\Theta)} \frac{\partial \log f(\mathbf{x} \mid \Theta)}{\partial \theta} \\
=& \sum_{\mathbf{x}} p(\mathbf{x} \mid \Theta) \frac{\partial \log f(\mathbf{x} \mid \Theta)}{\partial \theta} \\
=& \left\langle \frac{\partial \log f(\mathbf{c} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{C} \sim p(\mathbf{x} \mid \Theta)},
\end{aligned} \tag{A.1}$$

where $< \cdot >_{\mathbf{X}}$ denotes the mean expectation of $\cdot$ given data set $\mathbf{X}$. The first term of the right-hand side is easy to get with the given data, $\mathbf{d} \in \mathbf{D}$, and the second term can be approximated by generating data samples $\mathbf{C}$ according to $p(\mathbf{x} \mid \Theta)$.

Applying the RBM model function in Equation 3.15 to Equation A.1, the derivative of the log-likelihood with respect to a parameter θ is as follows::

$$
\begin{aligned}
\frac{\partial \hat{l}(\Theta \mid \mathbf{D})}{\partial \theta} &= \left\langle \frac{\partial \log f(\mathbf{d} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{D}} - \left\langle \frac{\partial \log f(\mathbf{c} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{C} \sim p(\mathbf{x}|\Theta)} \\
&= \left\langle \frac{\partial \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\Theta)}}{\partial \theta} \right\rangle_{\mathbf{D}} - \left\langle \frac{\partial \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\Theta)}}{\partial \theta} \right\rangle_{\mathbf{C} \sim p(\mathbf{v}|\Theta)} \\
&= \left\langle \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h}|\Theta)}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\Theta)}} \cdot \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{D}} \\
&\quad - \left\langle \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h}|\Theta)}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\Theta)}} \cdot \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{C} \sim p(\mathbf{v}|\Theta)} \\
&= \left\langle \sum_{\mathbf{h}} \frac{p(\mathbf{v}, \mathbf{h} \mid \Theta)}{p(\mathbf{v} \mid \Theta)} \cdot \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{D}} \\
&\quad - \left\langle \sum_{\mathbf{h}} \frac{p(\mathbf{v}, \mathbf{h} \mid \Theta)}{p(\mathbf{v} \mid \Theta)} \cdot \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{C} \sim p(\mathbf{v}|\Theta)} \\
&= \left\langle \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}, \Theta) \cdot \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{D}} \\
&\quad - \left\langle \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}, \Theta) \cdot \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{C} \sim p(\mathbf{v}|\Theta)} \\
&= \left\langle \left\langle \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{D_h} \sim p(\mathbf{h}|\mathbf{v},\Theta)} \right\rangle_{\mathbf{D}} \\
&\quad - \left\langle \left\langle \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\mathbf{C_h} \sim p(\mathbf{h}|\mathbf{c_v},\Theta)} \right\rangle_{\mathbf{C_v} \sim p(\mathbf{v}|\Theta)} \\
&= \left\langle \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\{\mathbf{D}, \mathbf{D}_h \sim p(\mathbf{h}|\mathbf{v},\Theta)\}} - \left\langle \frac{\partial -E(\mathbf{v}, \mathbf{h} \mid \Theta)}{\partial \theta} \right\rangle_{\{\mathbf{C_v}, \mathbf{C_h}\} \sim p(\mathbf{v},\mathbf{h}|\Theta)}.
\end{aligned}
\tag{A.2}
$$

The first term of the right-hand side can be gathered from the given data, $\{\mathbf{D}, \mathbf{D}_h\}$, and the second term can be approximated by generating model data samples $\{\mathbf{C_v}, \mathbf{C_h}\}$.

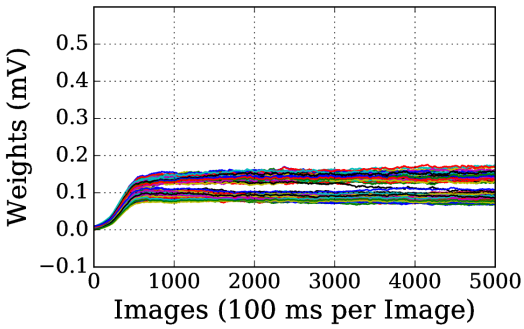
Appendix B

Detailed Experimental Results for Chapter 5

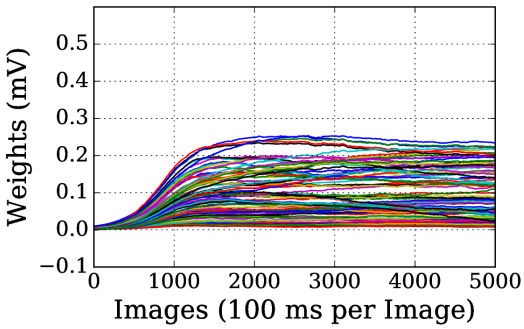
In Chapter 5, Figures 5.14 and 5.15 show the comparisons of different solutions for reducing correlations in training spiking Autoencoders (SAEs) and spiking Restricted Boltzmann Machines (SRBMs). In this section, Figures B.1 to B.8 present the detailed experimental results for this comparison.

To validate the learning capability of the proposed off-line training method on Leaky Integrate-and-Fire(LIF) neurons, Figures B.9 and B.10 demonstrate the results conducting the same experiments shown in Figures 5.14 and 5.15.

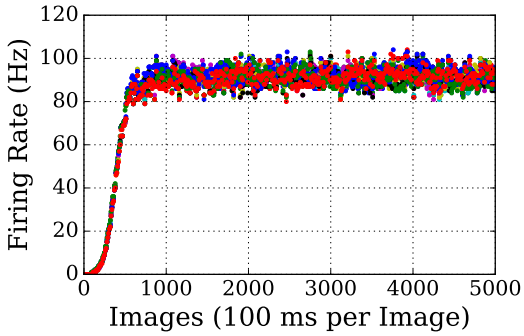
In addition, to provide a closer observation of the trained weights in Figures 5.17 and 5.18, the enlarged images are shown in Figures B.11 to B.22.



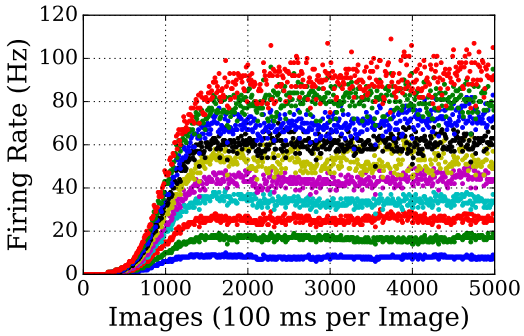
(a) Weights of Exp1



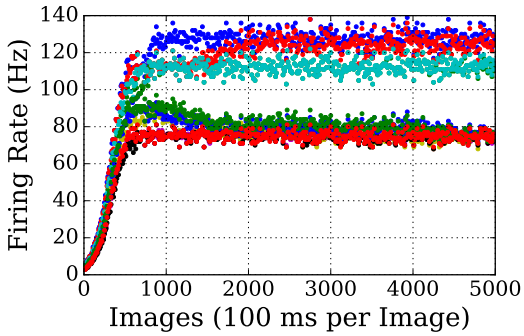
(b) Weights of Exp2



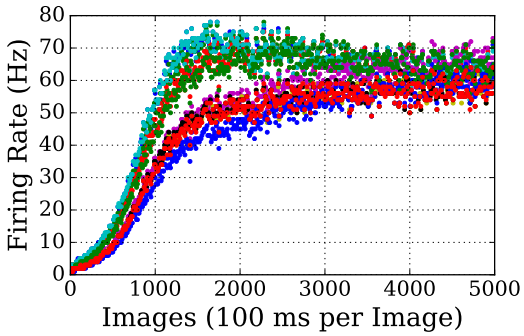
(c) Reconstruction of visible units in Exp1



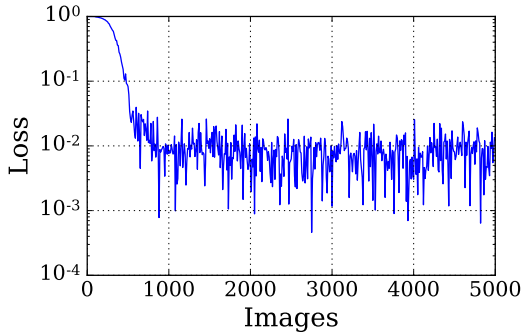
(d) Reconstruction of visible units in Exp2



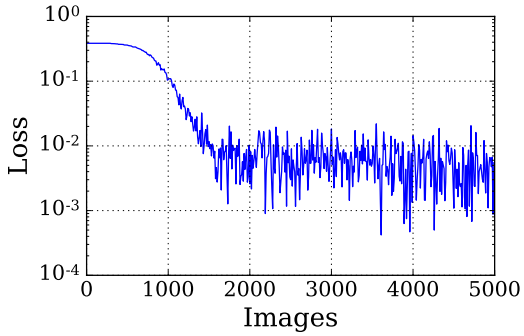
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

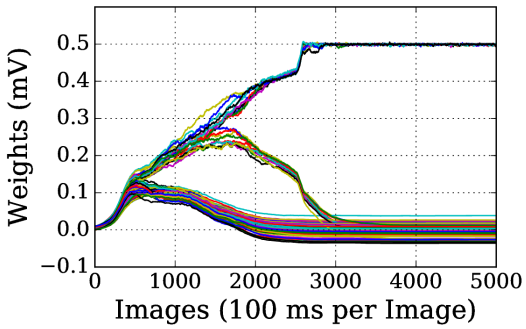


(g) Loss of Exp1

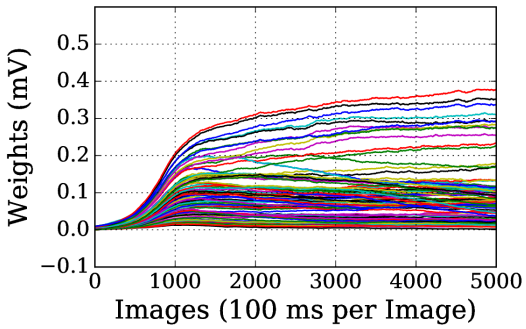


(h) Loss of Exp2

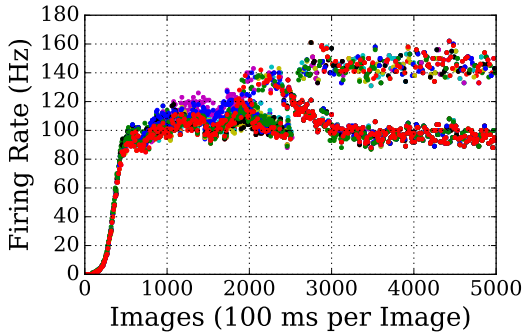
Figure B.1: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking AE with Solution 1. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



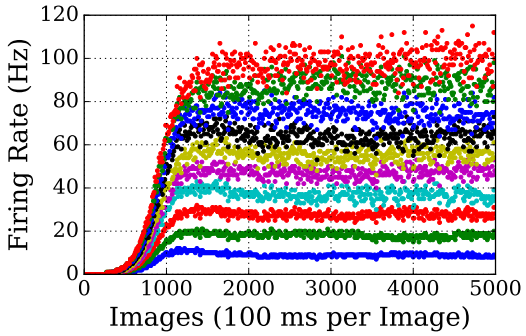
(a) Weights of Exp1



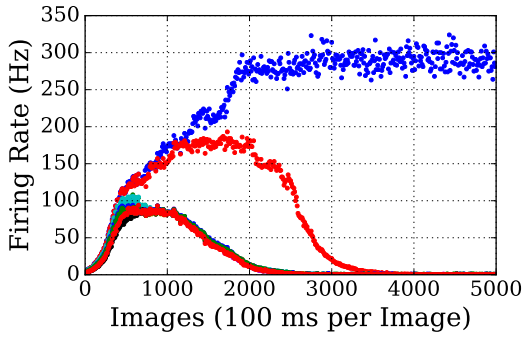
(b) Weights of Exp2



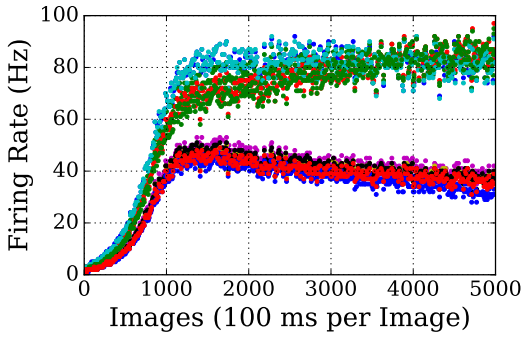
(c) Reconstruction of visible units in Exp1



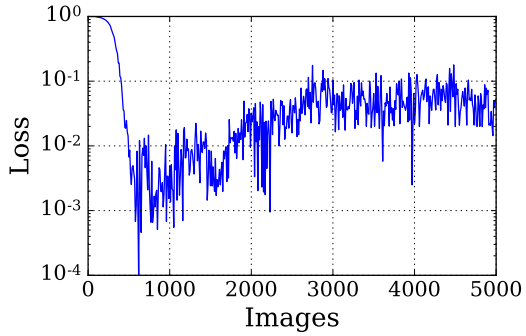
(d) Reconstruction of visible units in Exp2



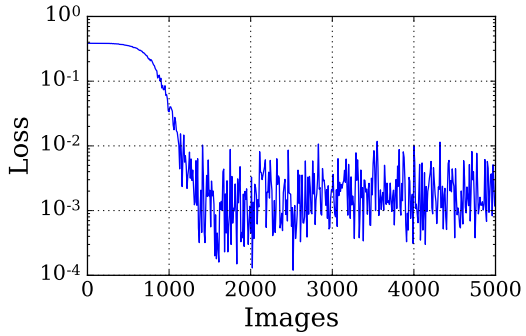
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

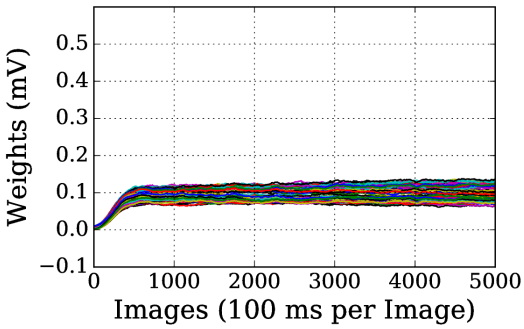


(g) Loss of Exp1

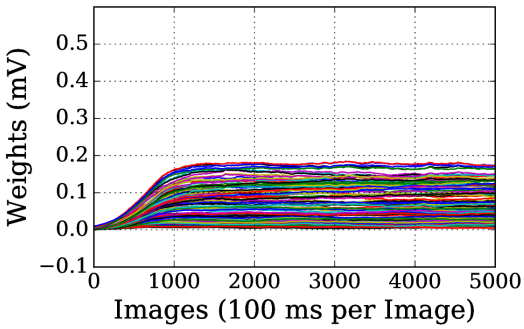


(h) Loss of Exp2

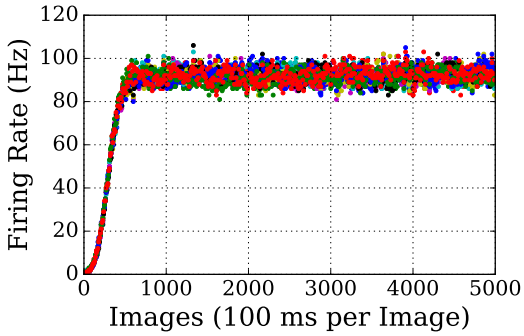
Figure B.2: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking RBM with Solution 1. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



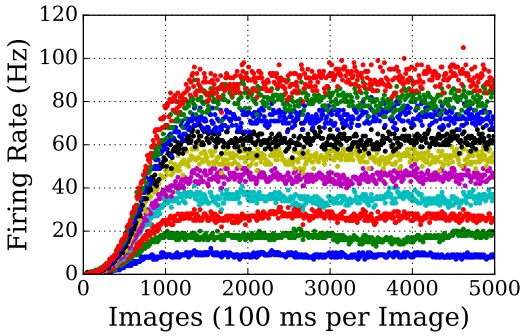
(a) Weights of Exp1



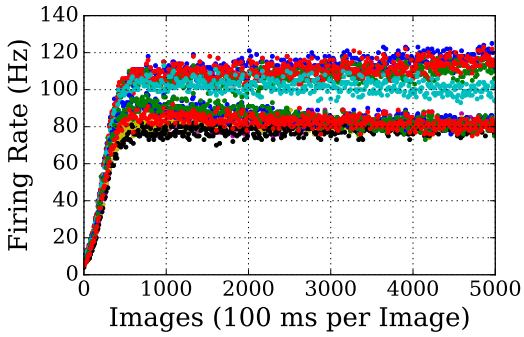
(b) Weights of Exp2



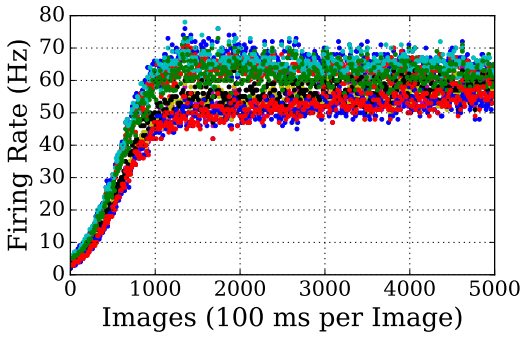
(c) Reconstruction of visible units in Exp1



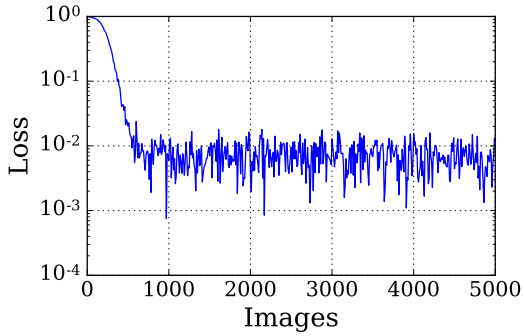
(d) Reconstruction of visible units in Exp2



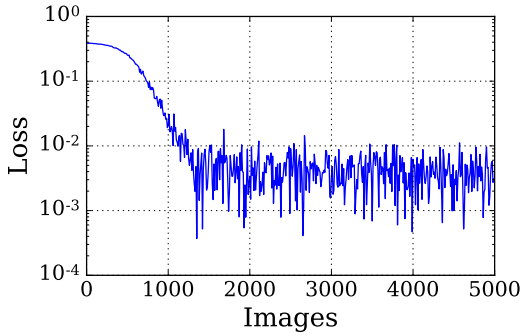
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

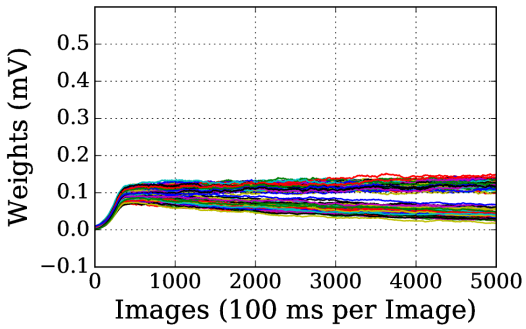


(g) Loss of Exp1

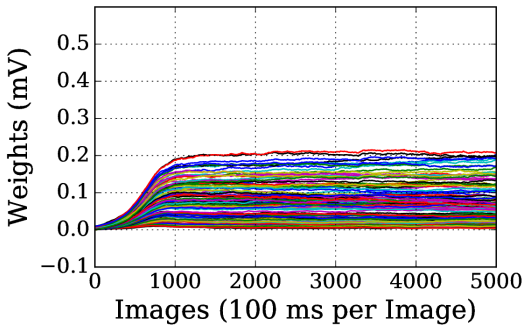


(h) Loss of Exp2

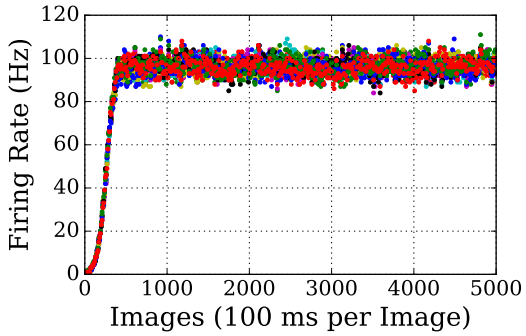
Figure B.3: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking AE with Solution 2. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



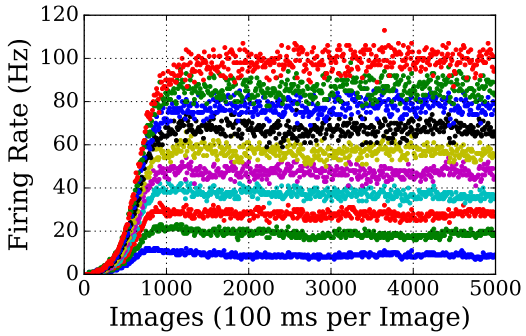
(a) Weights of Exp1



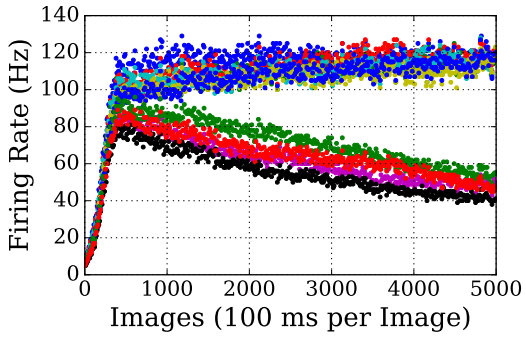
(b) Weights of Exp2



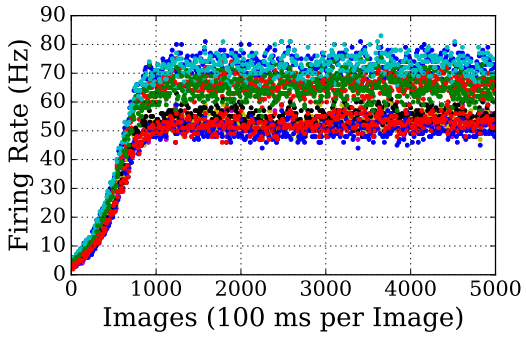
(c) Reconstruction of visible units in Exp1



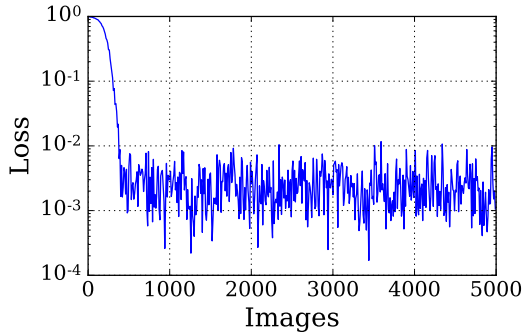
(d) Reconstruction of visible units in Exp2



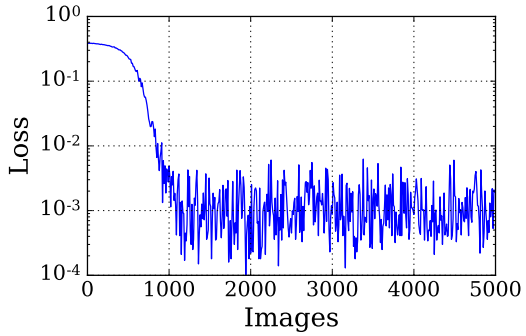
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

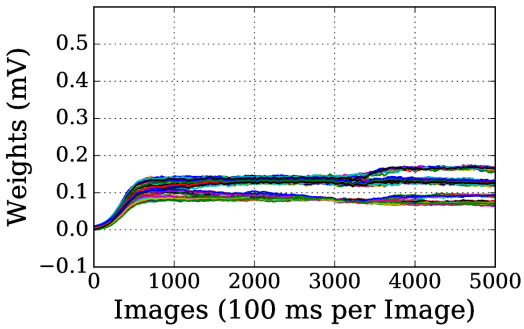


(g) Loss of Exp1

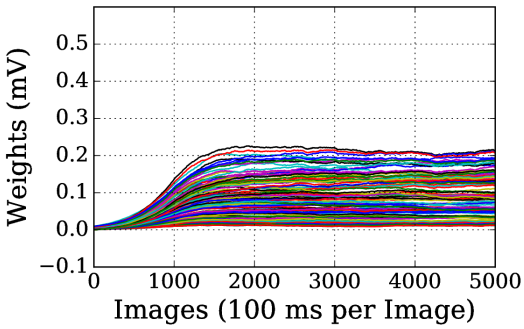


(h) Loss of Exp2

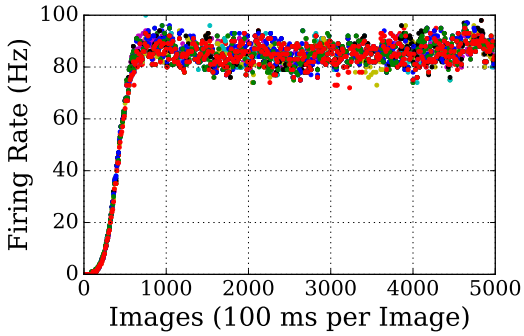
Figure B.4: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking RBM with Solution 2. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



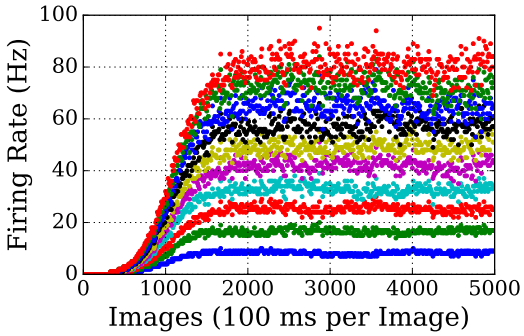
(a) Weights of Exp1



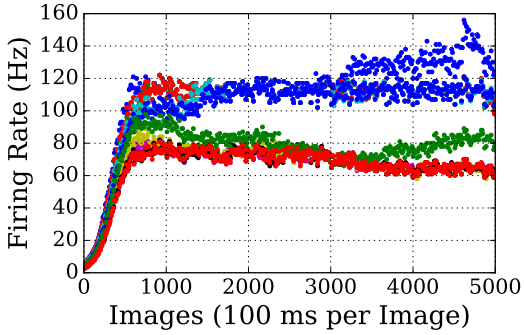
(b) Weights of Exp2



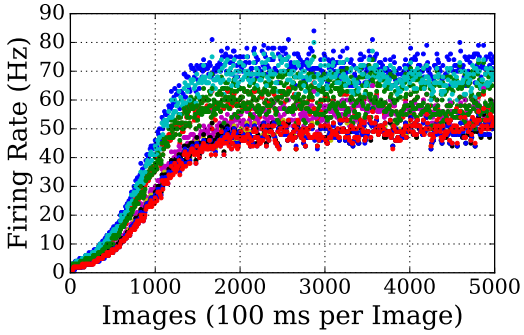
(c) Reconstruction of visible units in Exp1



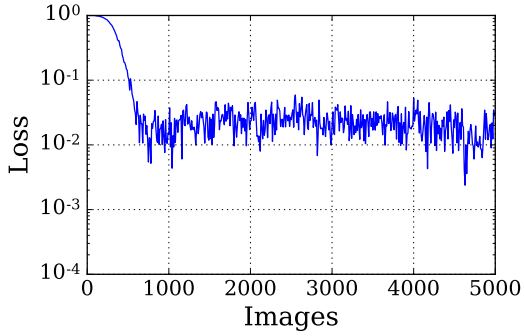
(d) Reconstruction of visible units in Exp2



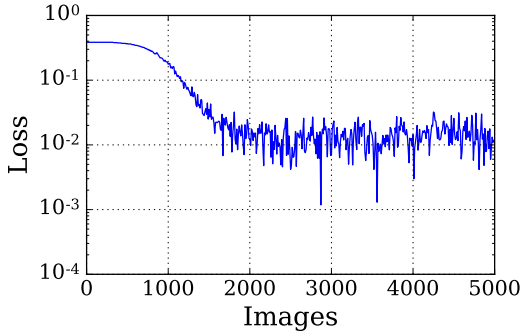
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

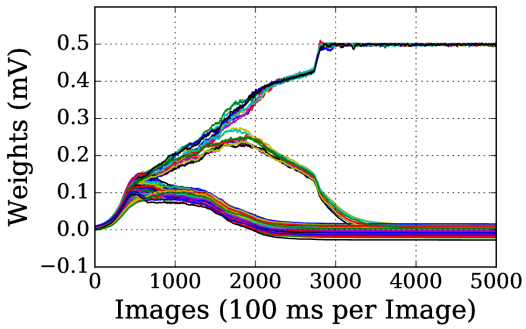


(g) Loss of Exp1

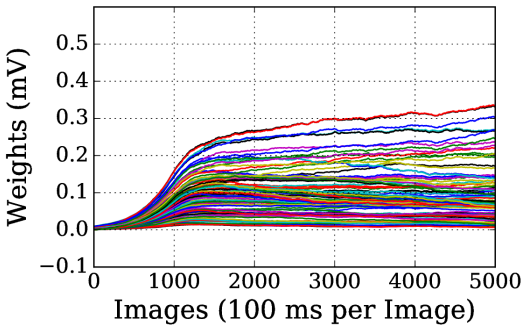


(h) Loss of Exp2

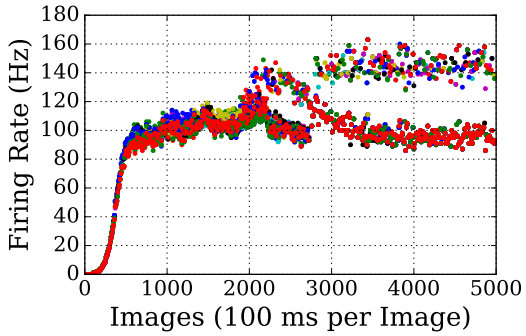
Figure B.5: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking AE with Solution 3. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



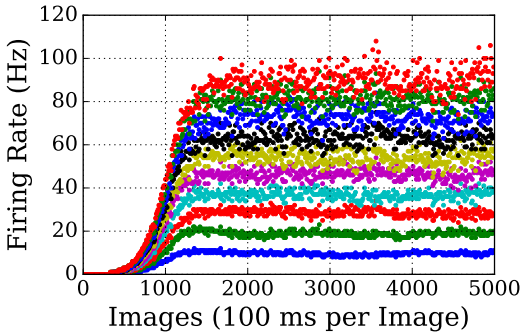
(a) Weights of Exp1



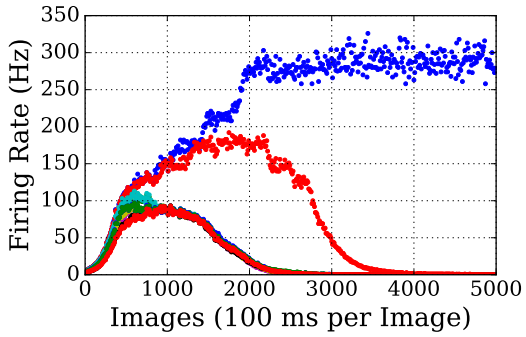
(b) Weights of Exp2



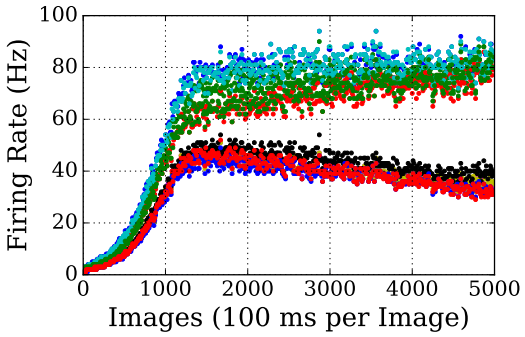
(c) Reconstruction of visible units in Exp1



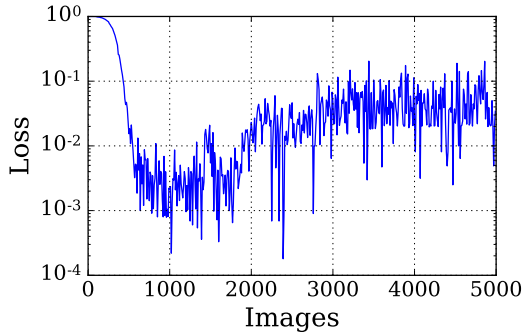
(d) Reconstruction of visible units in Exp2



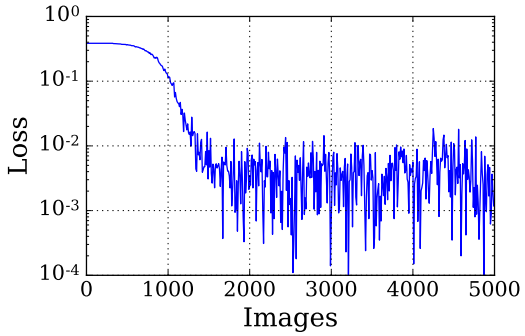
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

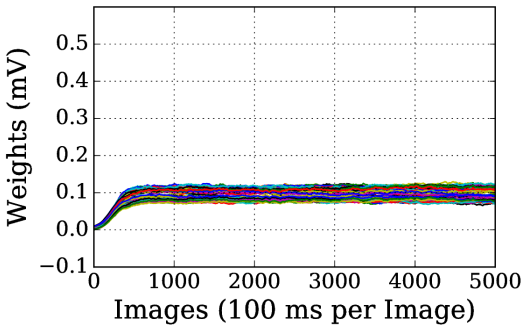


(g) Loss of Exp1

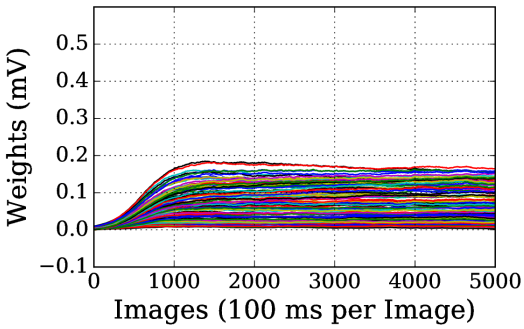


(h) Loss of Exp2

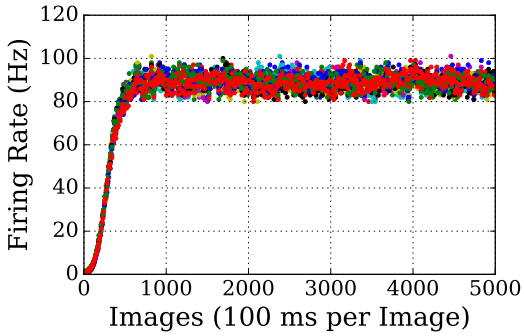
Figure B.6: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking RBM with Solution 3. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



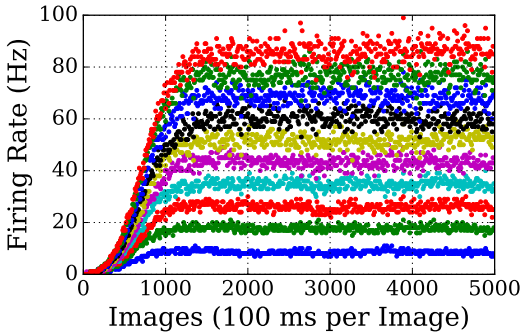
(a) Weights of Exp1



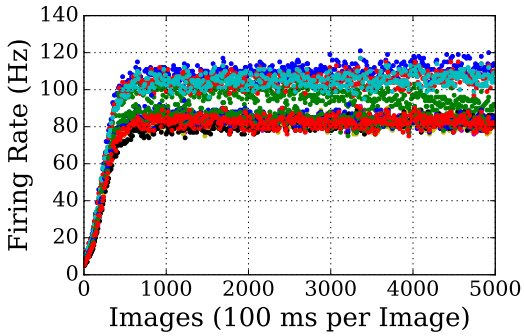
(b) Weights of Exp2



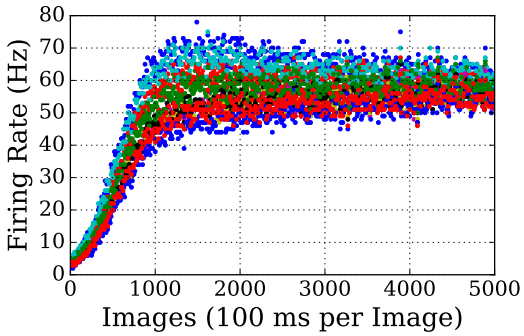
(c) Reconstruction of visible units in Exp1



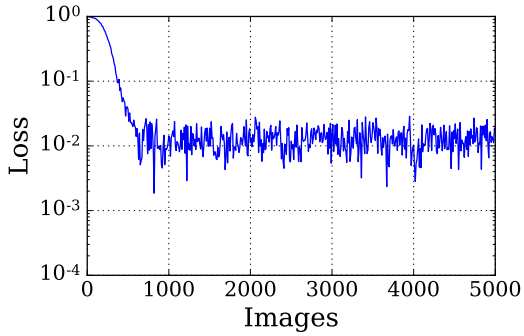
(d) Reconstruction of visible units in Exp2



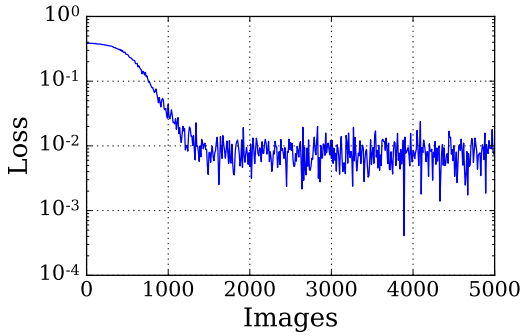
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

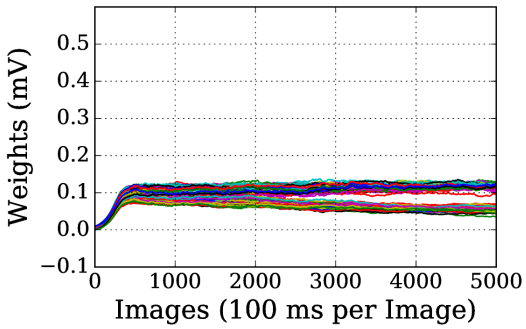


(g) Loss of Exp1

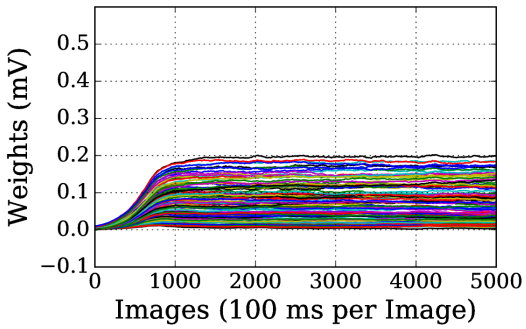


(h) Loss of Exp2

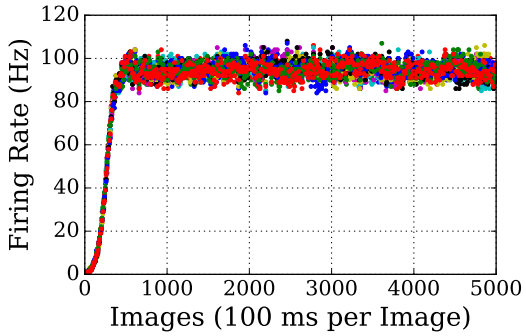
Figure B.7: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking AE with combined solutions. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



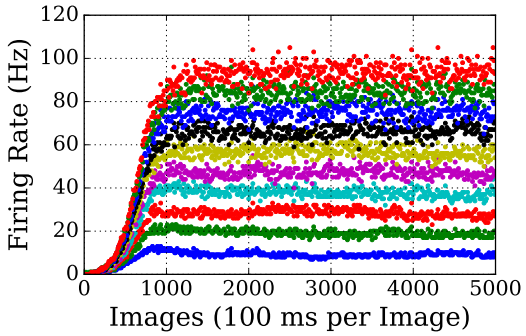
(a) Weights of Exp1



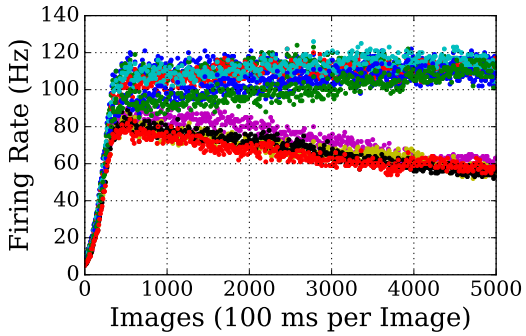
(b) Weights of Exp2



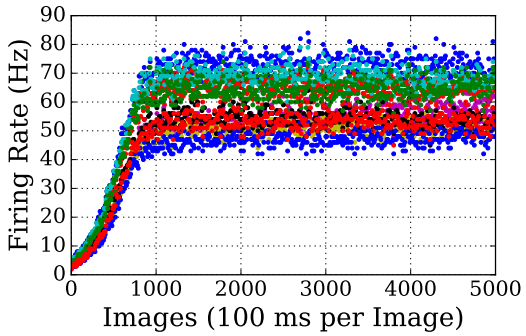
(c) Reconstruction of visible units in Exp1



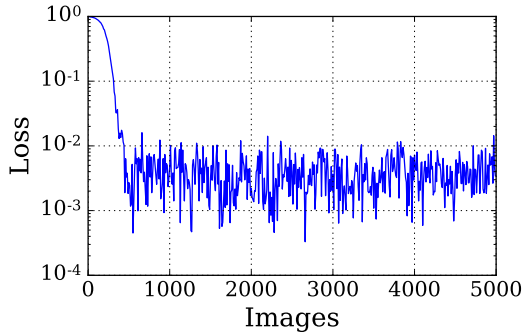
(d) Reconstruction of visible units in Exp2



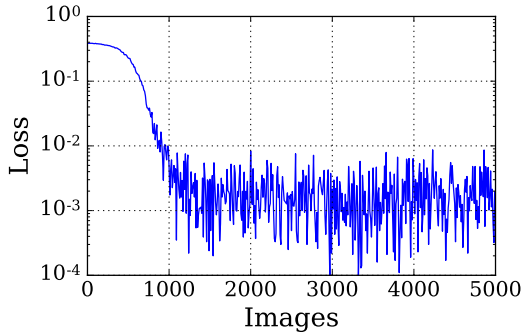
(e) Output of hidden units in Exp1



(f) Output of hidden units in Exp2

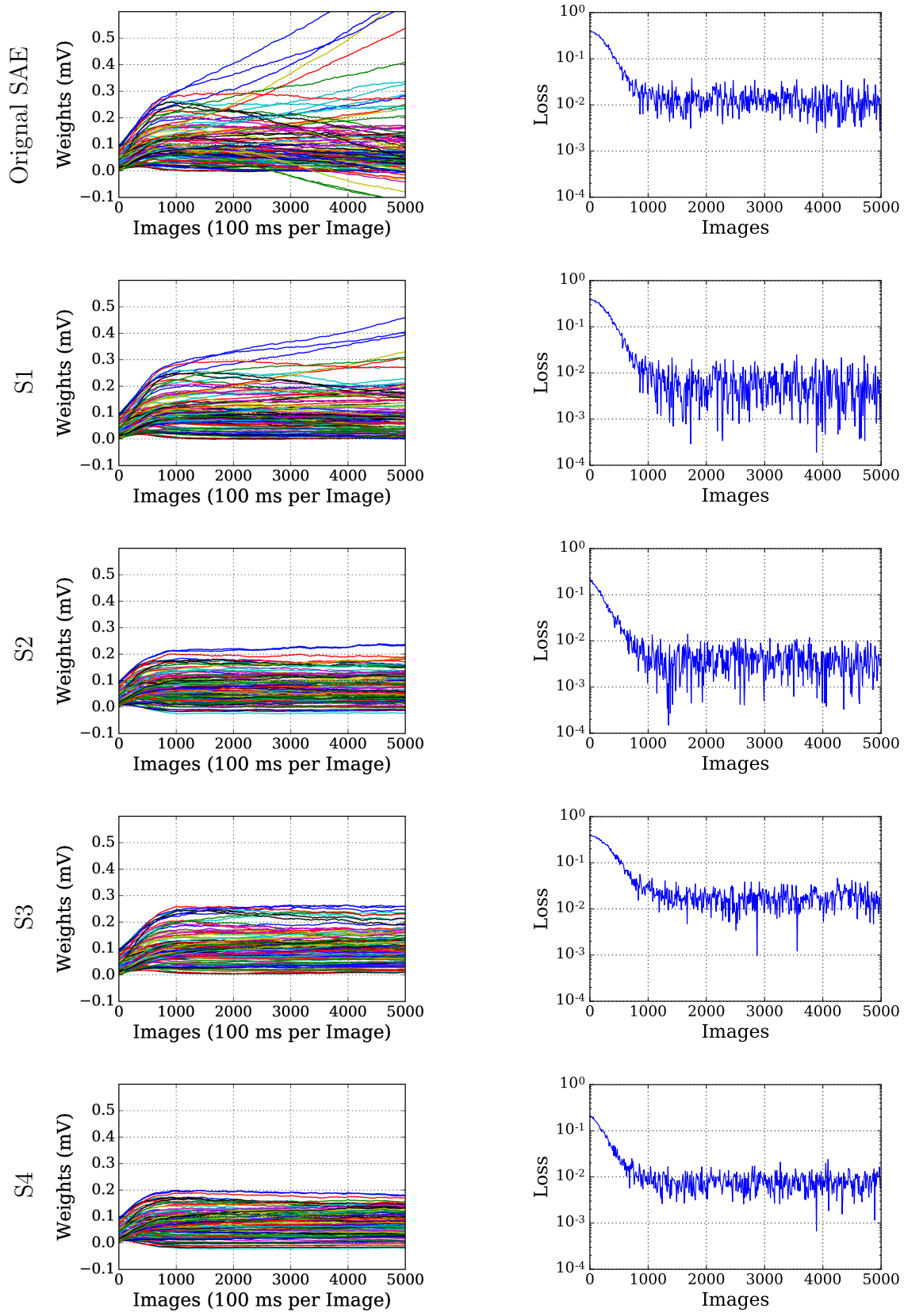


(g) Loss of Exp1



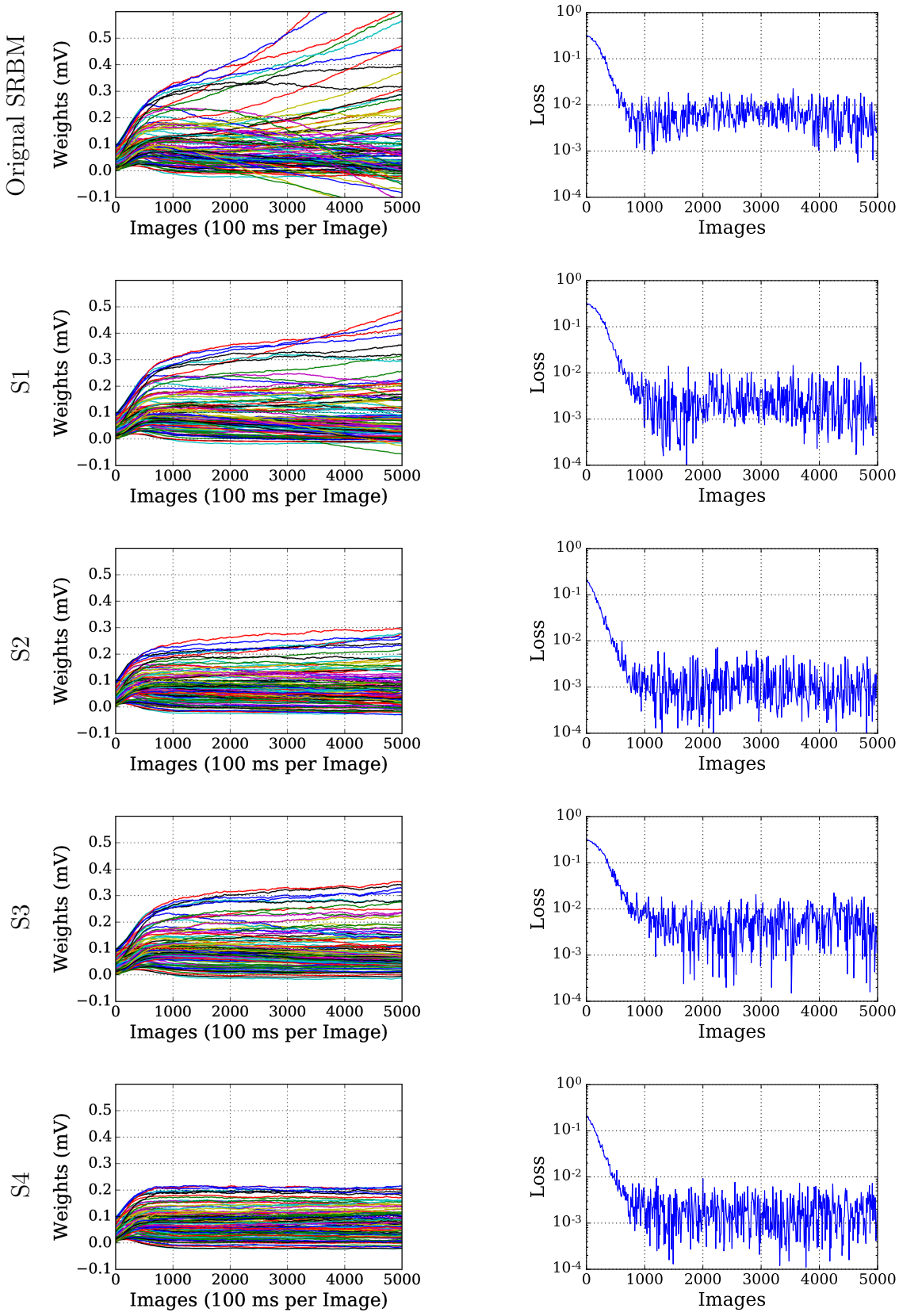
(h) Loss of Exp2

Figure B.8: Weights and firing rates of visible and hidden units change during training of the reconstruction tests of spiking RBM with combined solutions. Experiments 1) 10 visible units fully connected to 10 hidden units with Poisson spike trains of 100 Hz which lasted 100 ms; 2) same network fed with 10 Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



(a) Exp2 Weights (b) Exp2 Loss

Figure B.9: Comparisons of weights and loss of solutions for training SAE on Exp2 using LIF neurons: [S1] longer STDP window, [S2] noisy threshold, [S3] teaching signal, and [S4] combined solutions. 10 visible units fully connected to 10 hidden units with Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.



(a) Exp2 Weights

(b) Exp2 Loss

Figure B.10: Comparisons of weights and loss of solutions for training SRBM on Exp2 using LIF neurons: [S1] longer STDP window, [S2] noisy threshold, [S3] teaching signal, and [S4] combined solutions. 10 visible units fully connected to 10 hidden units with Poisson spike trains of firing rate ranging from 10 Hz to 100 Hz.

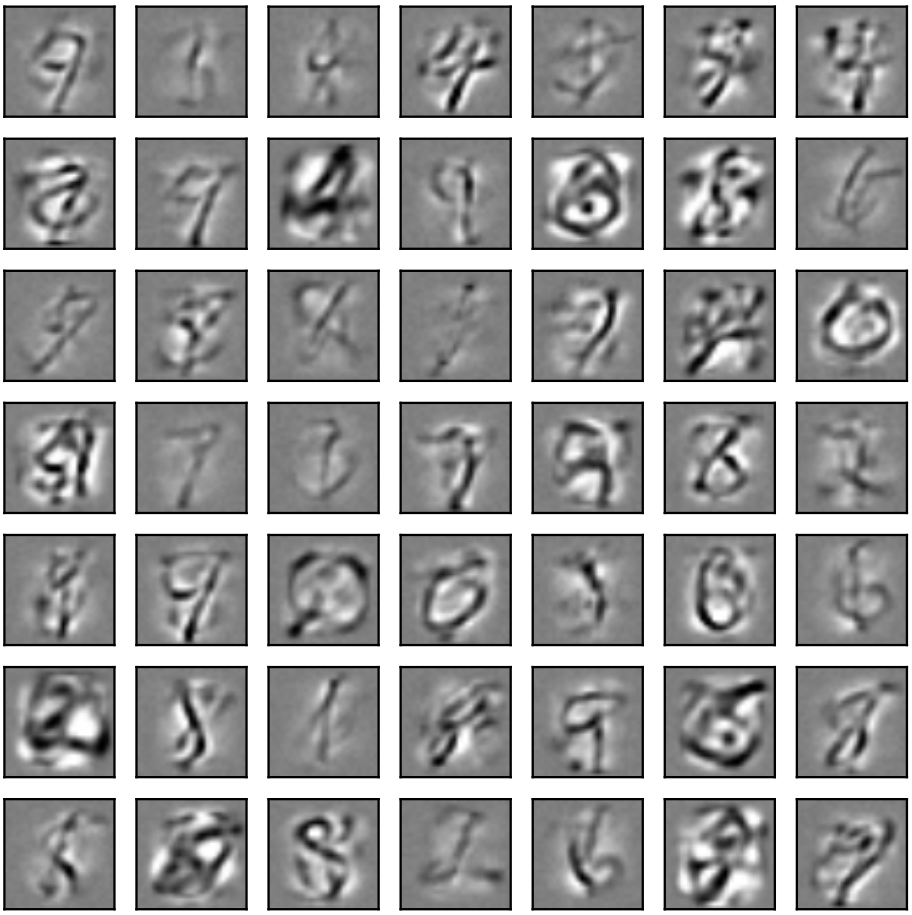


Figure B.11: Trained weights after 3 epochs of MNIST training using AE, same as Figure 5.17(a).

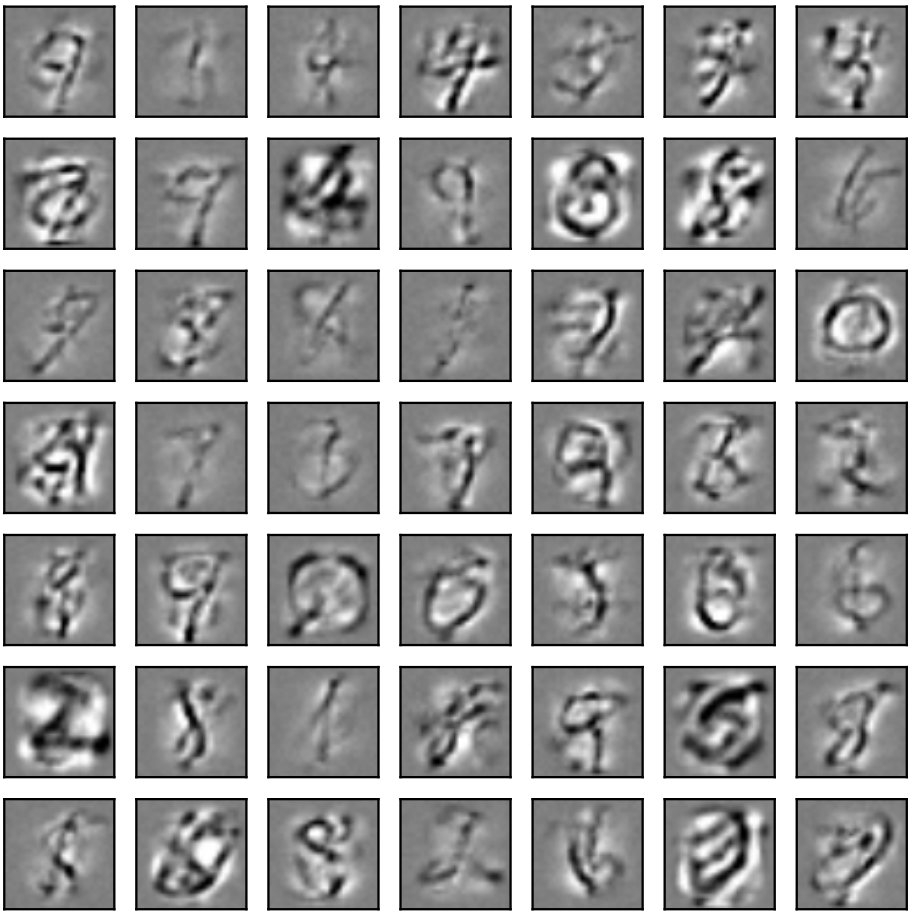


Figure B.12: Trained weights after 3 epochs of MNIST training using AE-NI, same as Figure 5.17(b).

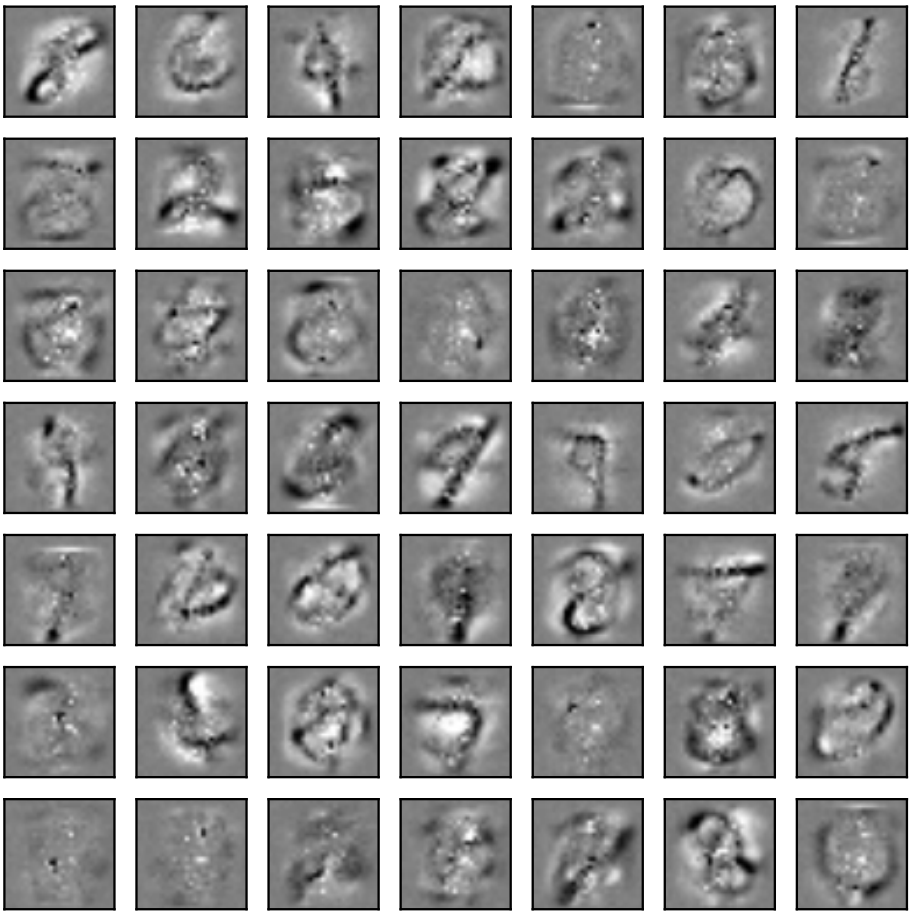


Figure B.13: Trained weights after 3 epochs of MNIST training using Original SAE, same as Figure 5.17(c).

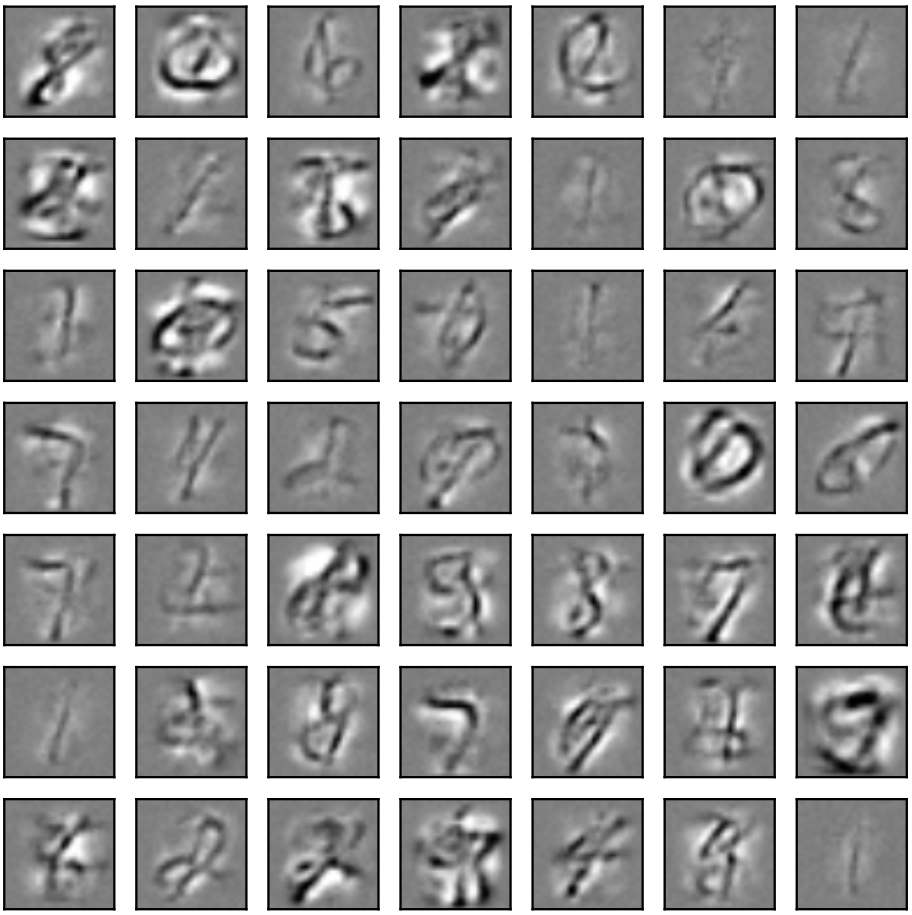


Figure B.14: Trained weights after 3 epochs of MNIST training using SAE-S2, same as Figure 5.17(d).

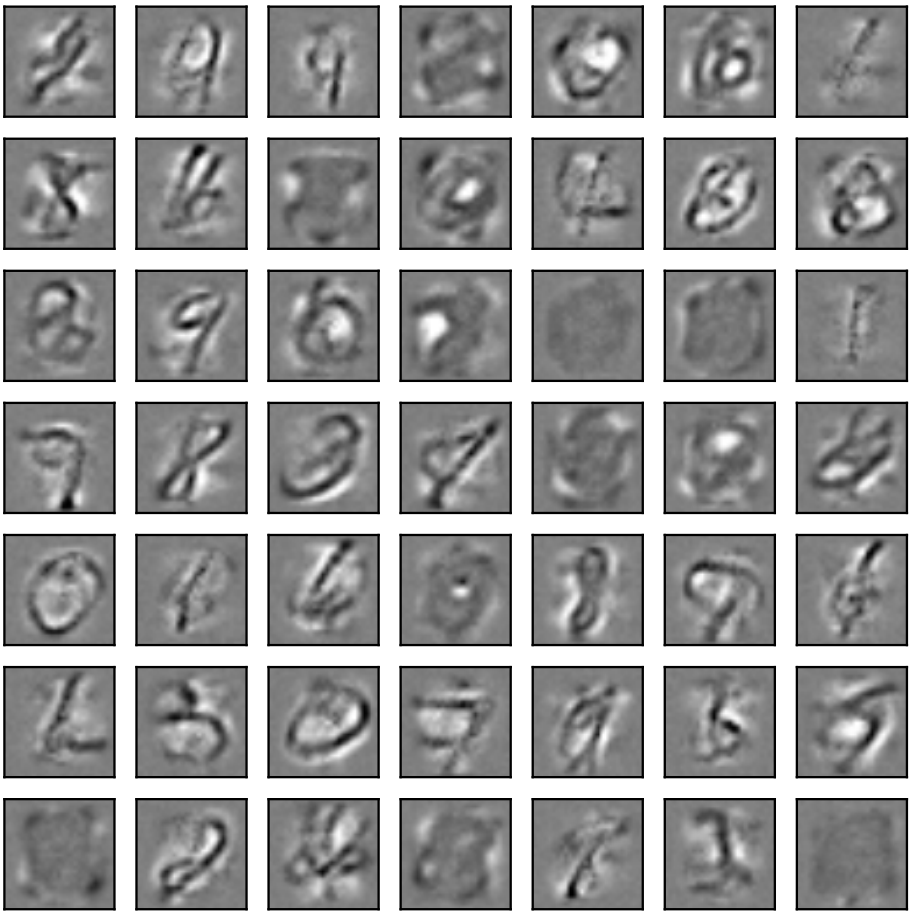


Figure B.15: Trained weights after 3 epochs of MNIST training using SAE-S3, same as Figure 5.17(e).

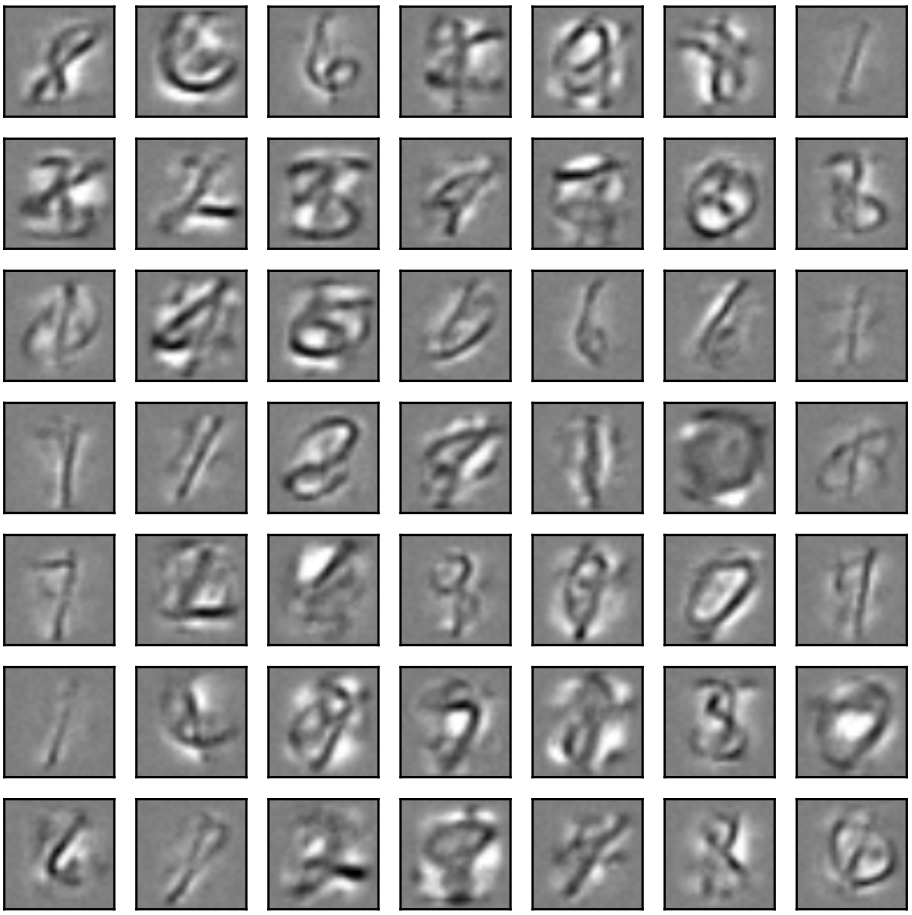


Figure B.16: Trained weights after 3 epochs of MNIST training using SAE-S4, same as Figure 5.17(f).

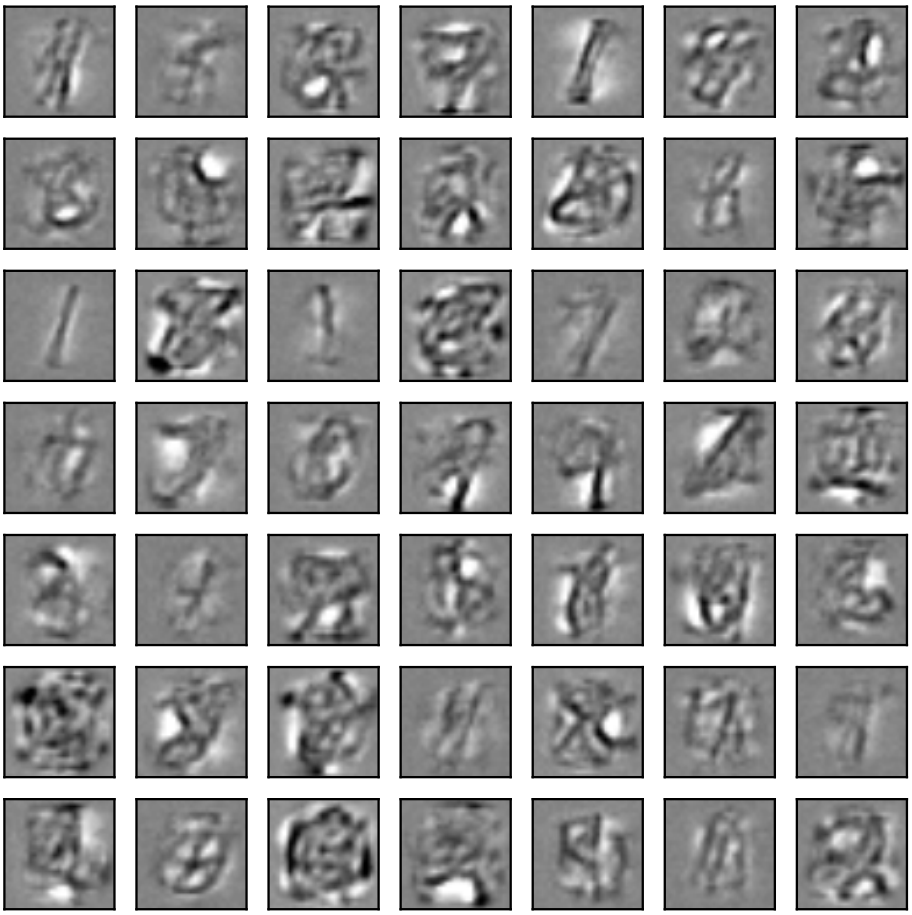


Figure B.17: Trained weights after 3 epochs of MNIST training using nRBM, same as Figure 5.18(a.)

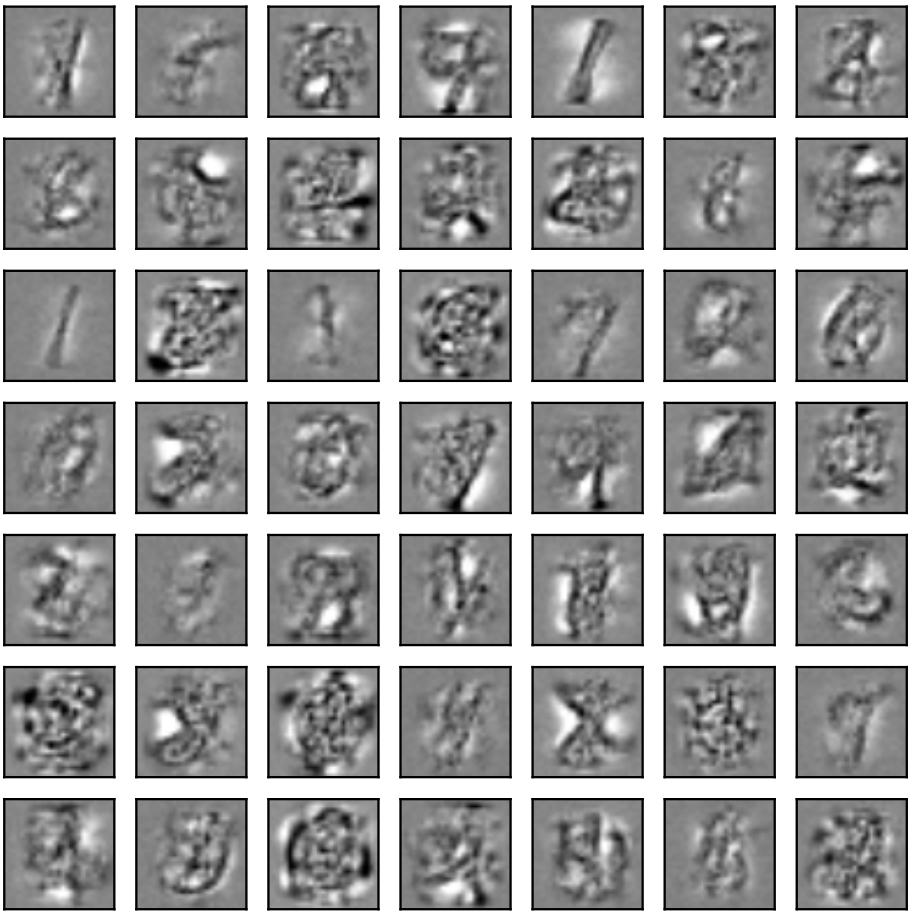


Figure B.18: Trained weights after 3 epochs of MNIST training using nRBM-NI, same as Figure 5.18(b).

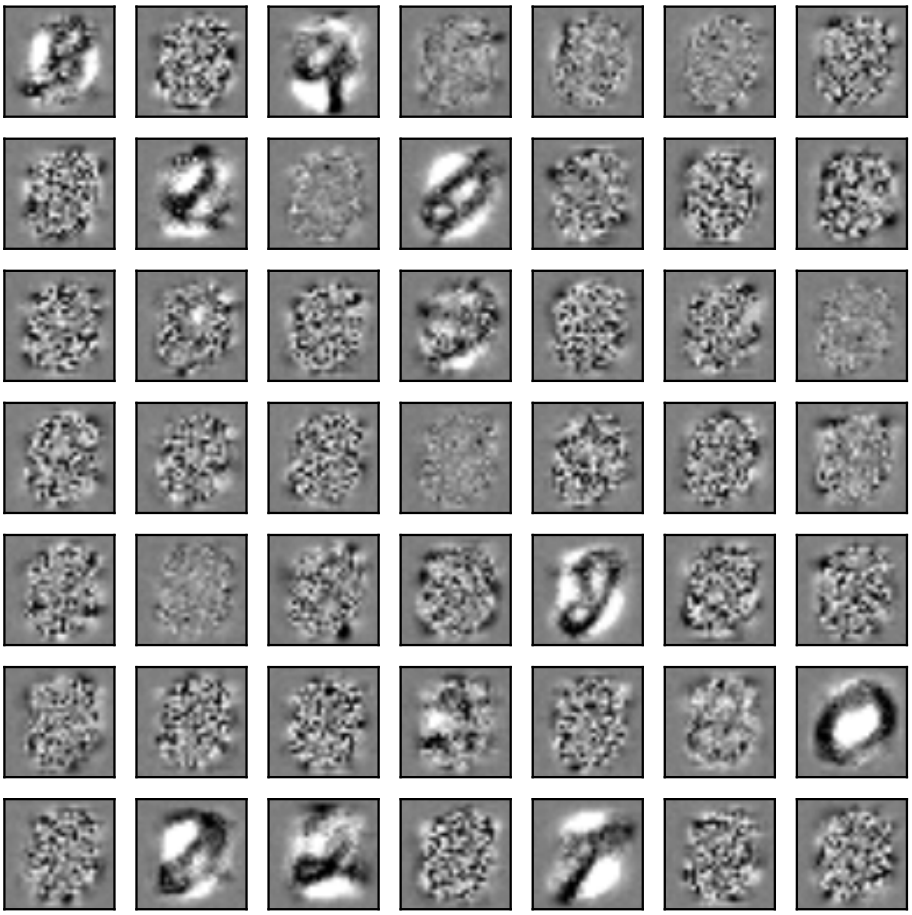


Figure B.19: Trained weights after 3 epochs of MNIST training using Original SRBM, same as Figure 5.18(c).

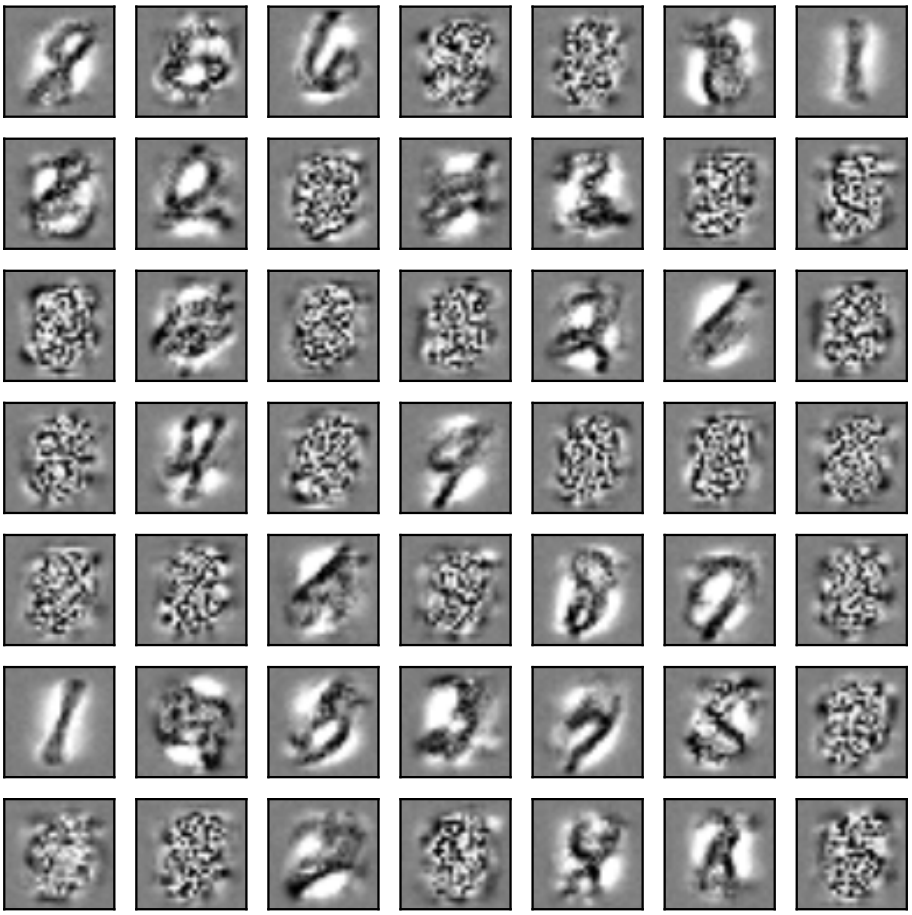


Figure B.20: Trained weights after 3 epochs of MNIST training using SRBM-S2, same as Figure 5.18(d).

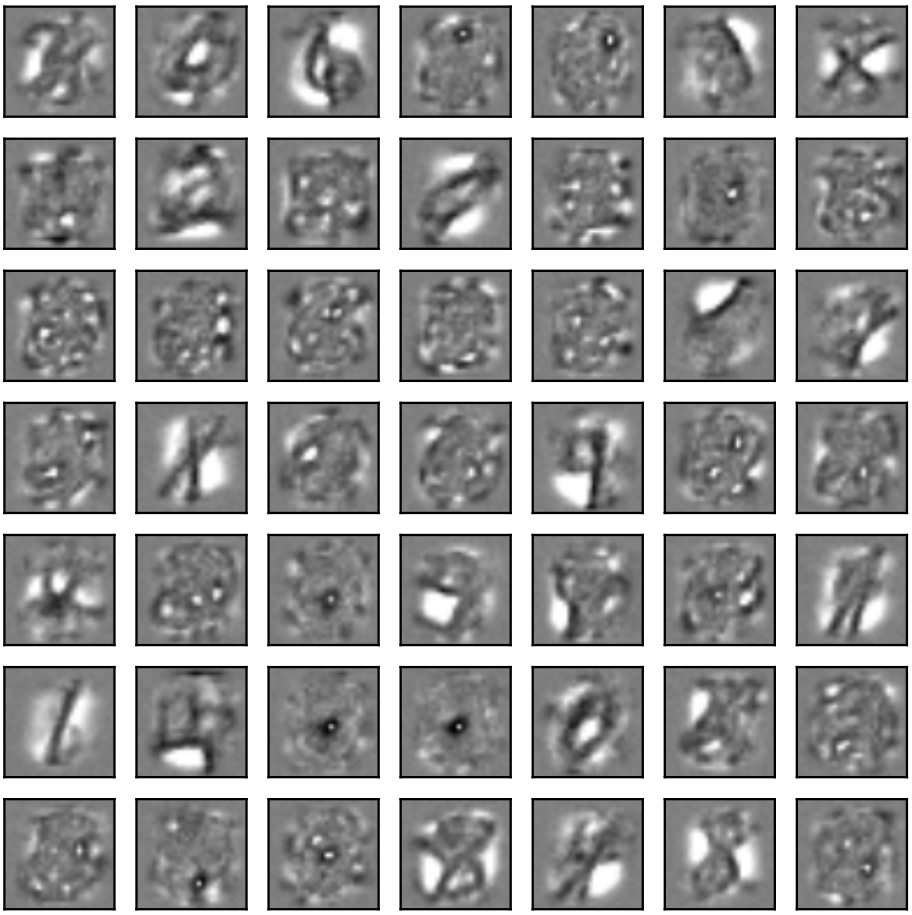


Figure B.21: Trained weights after 3 epochs of MNIST training using SRBM-S3, same as Figure 5.18(e).

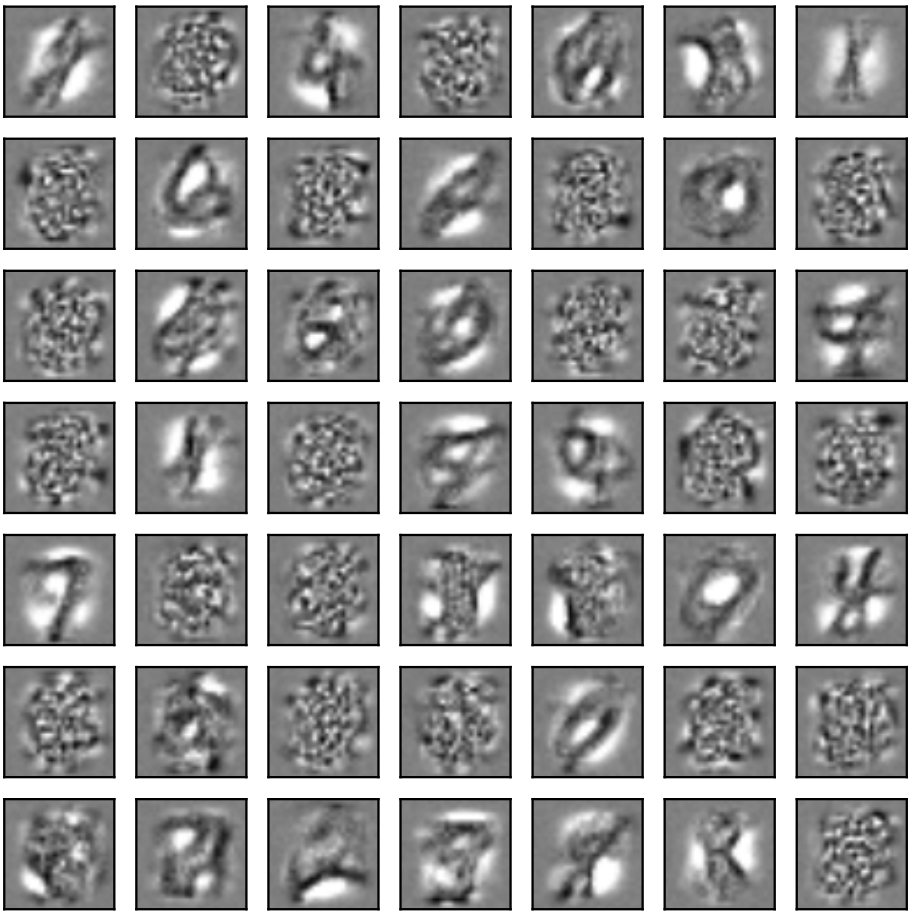


Figure B.22: Trained weights after 3 epochs of MNIST training using SRBM-S4, same as Figure 5.18(f).

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Ananthanarayanan, R., Esser, S. K., Simon, H. D., and Modha, D. S. (2009). The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12. IEEE.

Azevedo, F. A., Carvalho, L. R., Grinberg, L. T., Farfel, J. M., Ferretti, R. E., Leite, R. E., Lent, R., Herculano-Houzel, S., et al. (2009). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J.-M., Alvarez-Icaza, R., Arthur, J. V., Merolla, P., Boahen, K., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716.

Bi, G.-q. and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 18(24):10464–10472.

Bi, G.-q. and Poo, M.-m. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, 24(1):139–166.

Bichler, O., Querlioz, D., Thorpe, S. J., Bourgoin, J.-P., and Gamrat, C. (2012). Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Networks*, 32:339–348.

Bill, J. and Legenstein, R. (2014). A compound memristive synapse model for statistical learning through STDP in spiking neural networks. *Frontiers in neuroscience*, 8.

Blank, M., Gorelick, L., Shechtman, E., Irani, M., and Basri, R. (2005). Actions as space-time shapes. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1395–1402.

Bouvrie, J. (2006). Notes on convolutional neural networks.

Brader, J. M., Senn, W., and Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation*, 19(11):2881–2912.

Buesing, L., Bill, J., Nessler, B., and Maass, W. (2011). Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS Comput Biol*, 7(11):e1002211.

Burbank, K. S. (2015). Mirrored stdp implements autoencoder learning in a network of spiking neurons. *PLoS Comput Biol*, 11(12):e1004566.

Byrne, J. H., Heidelberger, R., and Waxham, M. N. (2014). *From molecules to networks: an introduction to cellular and molecular neuroscience*. Academic Press.

Camunas-Mesa, L., Zamarreño-Ramos, C., Linares-Barranco, A., Acosta-Jiménez, A. J., Serrano-Gotarredona, T., and Linares-Barranco, B. (2012). An event-driven multi-kernel convolution processor module for event-driven vision sensors. *Solid-State Circuits, IEEE Journal of*, 47(2):504–517.

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66.

Davison, A. P., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., and Yger, P. (2008). PyNN: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, 2.

De Garis, H., Shuo, C., Goertzel, B., and Ruiting, L. (2010). A world survey of artificial brain projects, Part I: Large-scale brain simulations. *Neurocomputing*, 74(1):3–29.

Deger, M., Helias, M., Boucsein, C., and Rotter, S. (2012). Statistical properties of superimposed stationary spike trains. *Journal of computational neuroscience*, 32(3):443–463.

Delbruck, T. (2008). Frame-free dynamic digital vision. In *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, pages 21–26.

Delorme, A. and Thorpe, S. J. (2001). Face identification using one spike per neuron: resistance to image degradations. *Neural Networks*, 14(6):795–803.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: a large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255.

DiCarlo, J. J., Zoccolan, D., and Rust, N. C. (2012). How does the brain solve visual object recognition? *Neuron*, 73(3):415–434.

Diehl, P. U., Cook, M., Tatsuno, M., and Song, S. (2015a). Unsupervised learning

of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015b). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE.

Diehl, P. U., Pedroni, B. U., Cassidy, A., Merolla, P., Neftci, E., and Zarrella, G. (2016a). Truehappiness: Neuromorphic emotion recognition on Truenorth. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 4278–4285. IEEE.

Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., and Neftci, E. (2016b). Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. *arXiv preprint*.

Drubach, D. (2000). *The brain explained*. Prentice Hall.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205.

Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V., and Modha, D. S. (2015). Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125.

Fabre-Thorpe, M., Richard, G., and Thorpe, S. J. (1998). Rapid categorization of natural images by rhesus monkeys. *Neuroreport*, 9(2):303–308.

Fischer, A. and Igel, C. (2012). An introduction to restricted Boltzmann machines. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36.

Fu, S.-Y., Yang, G.-S., and Kuai, X.-K. (2012). A spiking neural network based cortex-like mechanism and application to facial expression recognition. *Computational intelligence and neuroscience*, 2012:19.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.

Furber, S. and Temple, S. (2007). Neural systems engineering. *Journal of the Royal Society interface*, 4(13):193–206.

Furber, S. B., Galluppi, F., Temple, S., Plana, L., et al. (2014). The SpiNNaker Project. *Proceedings of the IEEE*, 102(5):652–665.

Garibaldi, P.-G., Camilleri, P., Liu, Q., and Furber, S. (2016). pyDVS: An extensible, real-time Dynamic Vision Sensor emulator using off-the-shelf hardware. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7.

Gautrais, J. and Thorpe, S. (1998). Rate coding versus temporal order coding: a theoretical approach. *Biosystems*, 48(1):57–65.

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

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal dynamics: from single neurons to networks and models of cognition*. Cambridge University Press.

Gewaltig, M.-O. and Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Goodman, D. and Brette, R. (2008). Brian: a simulator for spiking neural networks in Python. *Frontiers in neuroinformatics*, 2.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, volume 14, pages 1764–1772.

Gray, C. M. and Singer, W. (1989). Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. *Proceedings of the National Academy of Sciences*, 86(5):1698–1702.

Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

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

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hodgkin, A. L. and Huxley, A. F. (1939). Action potentials recorded from inside a nerve fibre. *Nature*, 144(3651):710–711.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of*

physiology, 117(4):500.

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

Hopkins, M. and Furber, S. (2015). Accuracy and efficiency in fixed-point neural ODE solvers. *Neural computation*, 27(10):2148–2182.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154.

Hunsberger, E. and Eliasmith, C. (2015). Spiking deep networks with LIF neurons. *arXiv preprint*.

Indiveri, G., Chicca, E., and Douglas, R. J. (2009). Artificial cognitive systems: from VLSI networks of spiking neurons to neuromorphic cognition. *Cognitive Computation*, 1(2):119–127.

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.-C., Dudek, P., Häfliger, P., Renaud, S., et al. (2011). Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5.

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

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070.

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral cortex*, 17(10):2443–2452.

Johansson, R. S. and Birznieks, I. (2004). First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature neuroscience*, 7(2):170–177.

Joubert, A., Belhadj, B., Temam, O., and Héliot, R. (2012). Hardware spiking neurons design: analog or digital? In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–5. IEEE.

Jug, F., Lengler, J., Krautz, C., and Steger, A. (2012). Spiking networks and their rate-based equivalents: does it make sense to use Siegert neurons? In *Swiss Soc. for Neuroscience*.

Karpathy, A. and Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137.

Keysers, C., Xiao, D.-K., Földiák, P., and Perrett, D. (2001). The speed of sight. *Journal of cognitive neuroscience*, 13(1):90–101.

Kiselev, I., Neil, D., and Liu, S.-C. (2016). Event-driven deep neural network hardware

system for sensor fusion. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pages 2495–2498. IEEE.

Kolb, H. (2003). How the retina works. *American scientist*, 91(1):28–35.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Krogh, A. and Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238.

La Camera, G., Giugliano, M., Senn, W., and Fusi, S. (2008). The response of cortical neurons to in vivo-like input current: theory and experiment. *Biological cybernetics*, 99(4-5):279–301.

Lagorce, X., Stromatias, E., Galluppi, F., Plana, L. A., Liu, S.-C., Furber, S. B., and Benosman, R. B. (2015). Breaking the millisecond barrier on SpiNNaker: implementing asynchronous event-based plastic models with ececond resolution. *Frontiers in Neuroscience*, 9:206.

Lazzaro, J. and Wawrzynek, J. (1995). A multi-sender asynchronous extension to the AER protocol. In *Advanced Research in VLSI, Conference on*, pages 158–158. IEEE Computer Society.

Le, Q. V. (2013). Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Huang, F. J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–97.

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

Linares-Barranco, B., Serrano-Gotarredona, T., and Serrano-Gotarredona, R. (2003). Compact low-power calibration mini-DACs for neural arrays with programmable weights. *Neural Networks, IEEE Transactions on*, 14(5):1207–1216.

Liu, J., Luo, J., and Shah, M. (2009). Recognizing realistic actions from videos "in the wild". In *Computer Vision and Pattern Recognition, 2009. CVPR. IEEE Conference on*, pages 1996–2003.

Liu, Q. and Furber, S. (2015). Real-time recognition of dynamic hand postures on a neuromorphic system. In *Artificial Neural Networks, 2015. ICANN. International*

Conference on, volume 1, page 979.

Liu, Q. and Furber, S. (2016). Noisy Softplus: a biology inspired activation function. In *International Conference on Neural Information Processing*, pages 405–412. Springer.

Liu, Q., Patterson, C., Furber, S., Huang, Z., Hou, Y., and Zhang, H. (2013). Modeling populations of spiking neurons for fine timing sound localization. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8. IEEE.

Liu, Q., Pineda-García, G., Stromatias, E., Serrano-Gotarredona, T., and Furber, S. B. (2016). Benchmarking spike-based visual recognition: a dataset and evaluation. *Frontiers in Neuroscience*, 10.

Liu, S.-C., van Schaik, A., Minch, B., and Delbruck, T. (2010). Event-based 64-channel binaural silicon cochlea with Q enhancement mechanisms. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2027–2030.

Lundqvist, D., Flykt, A., and Öhman, A. (1998). The Karolinska directed emotional faces (KDEF). *CD ROM from Department of Clinical Neuroscience, Psychology section, Karolinska Institutet*, (1998).

Lyons, M., Akamatsu, S., Kamachi, M., and Gyoba, J. (1998). Coding facial expressions with Gabor wavelets. In *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 200–205.

Mahowald, M. (1992). *VLSI analogs of neuronal visual processing: a synthesis of form and function*. PhD thesis, California Institute of Technology.

Marieb, E. N. and Hoehn, K. (2007). *Human anatomy & physiology*. Pearson Education.

Markram, H. (2006). The blue brain project. *Nature Reviews Neuroscience*, 7(2):153–160.

Masmoudi, K., Antonini, M., Kornprobst, P., and Perrinet, L. (2010). A novel bio-inspired static image compression scheme for noisy data transmission over low-bandwidth channels. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 3506–3509.

Matsugu, M., Mori, K., Ishii, M., and Mitarai, Y. (2002). Convolutional spiking neural network model for robust face detection. In *Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on*, volume 2, pages 660–664.

Mead, C. (1989). *Analog VLSI and neural systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2017). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems*, PP(99):1–17.

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biological cybernetics*, 98(6):459–478.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814.

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., and Cauwenberghs, G. (2013). Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in neuroscience*, 7.

Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017). Event-driven random back-propagation: enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, 11.

Neftci, E. O., Pedroni, B. U., Joshi, S., Al-Shedivat, M., and Cauwenberghs, G. (2016). Stochastic synapses enable efficient brain-inspired learning machines. *Frontiers in Neuroscience*, 10:241.

Neil, D. (2013). Online learning in event-based restricted Boltzmann machines. Master's thesis, Institute of Neuroinformatics, ETH Zurich.

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

Nessler, B., Pfeiffer, M., Buesing, L., and Maass, W. (2013). Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Comput Biol*.

O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T., and Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in neuroscience*, 7.

O'Connor, P. and Welling, M. (2016). Deep spiking networks. *arXiv preprint arXiv:1602.08323*.

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9(437).

Park, J., Ha, S., Yu, T., Neftci, E., and Cauwenberghs, G. (2014). A 65K-neuron 73-Mevents/s 22-pJ/event asynchronous micro-pipelined integrate-and-fire array transceiver. In *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*,

pages 675–678. IEEE.

Pedram, M. and Nazarian, S. (2006). Thermal modeling, analysis, and management in VLSI circuits: principles and methods. *Proceedings of the IEEE*, 94(8):1487–1501.

Petrovici, M. A., Bill, J., Bytschok, I., Schemmel, J., and Meier, K. (2013). Stochastic inference with deterministic spiking neurons. *arXiv preprint arXiv:1311.3211*.

Ponulak, F. and Kasinski, A. (2010). Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Computation*, 22(2):467–510.

Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B., and Delbruck, T. (2014). Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. *Proceedings of the IEEE*, 102(10):1470–1484.

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., and Indiveri, G. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Rauch, A., La Camera, G., Lüscher, H.-R., Senn, W., and Fusi, S. (2003). Neocortical pyramidal cells respond as integrate-and-fire neurons to in vivo-like input currents. *Journal of neurophysiology*, 90(3):1598–1612.

Reece, J. B., Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V., Jackson, R. B., et al. (2011). *Campbell biology*. Pearson Boston.

Riesenhuber, M. and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Samadi, A., Lillicrap, T. P., and Tweed, D. B. (2017). Deep learning with dynamic spiking neurons and fixed feedback weights. *Neural computation*.

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

Schüldt, C., Laptev, I., and Caputo, B. (2004). Recognizing human actions: a local SVM approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE.

Sen, B. and Furber, S. (2009). Evaluating rank-order code performance using a biologically-derived retinal model. In *Neural Networks, 2009. IJCNN. International*

Joint Conference on, pages 2867–2874. IEEE.

Serrano-Gotarredona, T. and Linares-Barranco, B. (2013). A 128×128 1.5% contrast sensitivity 0.9% FPN 3μs latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *Solid-State Circuits, IEEE Journal of*, 48(3):827–838.

Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T. (2007). Robust object recognition with cortex-like mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):411–426.

Sharp, T., Galluppi, F., Rast, A., and Furber, S. (2012). Power-efficient simulation of detailed cortical microcircuits on SpiNNaker. *Journal of neuroscience methods*, 210(1):110–118.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926.

Squire, L. R. and Kosslyn, S. M. (1998). *Findings and current opinion in cognitive neuroscience*. MIT Press.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

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

Stromatias, E., Neil, D., Galluppi, F., Pfeiffer, M., Liu, S.-C., and Furber, S. (2015a). Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on SpiNNaker. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE.

Stromatias, E., Neil, D., Pfeiffer, M., Galluppi, F., Furber, S. B., and Liu, S.-C. (2015b). Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in neuroscience*, 9.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with

neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Tan, C., Lallee, S., and Orchard, G. (2015). Benchmarking neuromorphic vision: lessons learnt from computer vision. *Frontiers in Neuroscience*, 9(374).

Van Rullen, R. and Thorpe, S. J. (2001). Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. *Neural computation*, 13(6):1255–1283.

Van Rullen, R. and Thorpe, S. J. (2002). Surfing a spike wave down the ventral stream. *Vision research*, 42(23):2593–2615.

Von Der Malsburg, C. (1994). The correlation theory of brain function. In *Models of neural networks*, pages 95–119. Springer.

Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Widrow, B. and Hoff, M. E. (1960). *Adaptive switching circuits*.

Xie, J., Xu, L., and Chen, E. (2012). Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems*, pages 341–349.

Xu, L. (1993). Least mean square error reconstruction principle for self-organizing neural-nets. *Neural networks*, 6(5):627–648.

Yang, M., Liu, S.-C., and Delbruck, T. (2015). A dynamic vision sensor with 1% temporal contrast sensitivity and in-pixel asynchronous delta modulator for event encoding. *Solid-State Circuits, IEEE Journal of*, 50(9):2149–2160.

Yarbus, A. L. (1967). *Eye movements during perception of complex objects*. Springer.

Yu, T., Park, J., Joshi, S., Maier, C., and Cauwenberghs, G. (2012). 65K-neuron integrate-and-fire array transceiver with address-event reconfigurable synaptic routing. In *Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE*, pages 21–24.

Zhao, B., Ding, R., Chen, S., Linares-Barranco, B., and Tang, H. (2015). Feedforward categorization on AER motion events using cortex-like features in a spiking neural network. *Neural Networks and Learning Systems, IEEE Transactions on*, 26(9):1963–1978.