

Imperial College London
Department of Electrical and Electronic Engineering

Robust and Efficient Training on Deep Spiking Neural Networks

Nicolas Perez-Nieves

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy at Imperial College London, January 2023

Statement of originality

I declare that the content of this thesis, and all the work herein detailed, is originated by myself. All the work that is not my own is appropriately referenced and accredited. Main ideas and some figures have been published in the following papers, where I have been involved as the first author:

- Perez-Nieves, Nicolas, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Advantages of heterogeneity of parameters in spiking neural network training. *Cognitive Computational Neuroscience (CCN)*, 2019
- Perez-Nieves, Nicolas, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Neural heterogeneity promotes robust learning. *Nature communications*, 2021
- Perez-Nieves, Nicolas, and Dan Goodman. Sparse spiking gradient descent. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021

Copyright declaration

The copyright of this manuscript rests with the author. The manuscript's contents are licensed under the [Creative Commons Attribution Non-Commercial No Derivatives 4.0 International](#) license (CC BY-NC-ND 4.0).

Under this license, you are free to copy and distribute the material in any medium or format under the conditions that you credit the author and do not use it for commercial purposes. Moreover, if you remix, transform, or build upon the material, you may not distribute modified versions of this work.

The license may not give you all the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Seek permission from the author for uses of the content of this thesis that are not included in the license.

Abstract

This thesis focuses on the study of training deep spiking neural networks (SNNs). In recent years there has been an increasing interest in using spiking neurons for deep learning with the aim of leveraging their unique properties and characteristics. These include their potential for very energy efficient training and inference due to their highly sparse activity and their suitability to model biological neurons.

We first introduce the fundamental models that are used in the SNN training literature, including neuron models at different levels of abstraction, synapses and networks of neurons and neurons under noise. We also review the main SNN training methods that have been developed to this date. Then, we study the role of neural heterogeneity and study the performance and robustness of SNNs under different heterogeneity schemes on two different supervised learning methods. Next, we show how the sparse activity present in the forward pass on SNNs can also be achieved in the backward pass, leading to highly efficient implementations that can speed up the backward pass up to 150x and save 85% of the memory. Finally, we aim to solve the weight initialisation problem for SNNs to achieve a predictable network activity as well as prevent the gradient from vanishing or exploding. In this process, we identify and solve the firing rate collapse issue caused by the discretisation of SNNs for simulation. In addition, we obtain theoretical and empirical results for a general SNN initialisation strategy making use of variance propagation and diffusion/shot-noise/threshold integration methods, as well as the solution to the firing rate collapse problem we previously found.

Besides the ideas and experiments discussed in this thesis, code for the methods described here can be found in <https://github.com/npvoid>.

Acknowledgements

First, I would like to thank my supervisor, Dan, for providing me with excellent guidance, support, and encouragement throughout my PhD studies. I am specially grateful for your enthusiasm and willingness to extend our meetings way beyond the scheduled time so that we could solve the problem at hand and for giving me room to pursue my own ideas and research interests.

I am also grateful to the past and present Neural Reckoning members for their interest and support. A special mention to Arnaud, Marcus and Gabriel for their feedback and advice.

I would also like to thank Axel and Adrián for their insights into the world of deep learning and specially for all the fun and laughs we had in the lab, making the whole PhD experience more enjoyable.

Special thanks to my old friend from high school, Pablo, with whom I reunited again at Imperial College after so many years and for his invaluable advice on stochastic calculus.

Gracias a mis padres, Esteban y Alicia y a mi hermana Marina por estar siempre dispuestos a ayudarme y hacer más fácil cumplir mis sueños. Gracias por apoyarme desde el primer momento en todas las etapas que me han llevado hasta aquí.

Muchas gracias también a mis abuelos Bienve, Abueli y Mario porque, aunque ya no estéis aquí, sé que estaríais orgullosos de dónde he llegado. Muchas gracias también a mi abuela Milagros por pensar en mí e intentar encontrar siempre cómo hacer que viva más cerca.

Me gustaría mencionar también a algunos profesores que han sido especialmente importantes para mí. A Begoña, que fue mi profesora de matemáticas en sexto de primaria, por transmitirnos la ilusión por aprender, por su sentido del humor y por su esfuerzo para hacer sentirse valorado a cada uno de sus alumnos. A Antonio, que fue mi profesor de lengua y literatura en bachillerato, a quién echo de menos

profundamente. Además de enseñarme a leer de verdad, Antonio me enseñó a vivir en equilibrio entre lo grave y trascendente, y lo insustancial e intrascendente de las cosas.

Quisiera mencionar también a todos los amigos que han estado ahí durante todo este tiempo y en especial a Paula, Ernes y Aitor porque han sido siempre los mejores amigos desde el primer momento.

Por último, gracias Alex por ser tan atenta y paciente conmigo, por compartir mis ilusiones, por tu amor, por tu sentido del humor y por hacer que todo sea más fácil. Sin ti, no habría podido sobrevivir todos los vaivenes del doctorado durante estos años.

To my fiancée, Alex

*Esta forma de alegría crece cuando ya
no es nada para el mundo, pero florece
cuando es reconocida por el mundo.*

This form of joy grows when it is no
longer anything to the world, but it flour-
ishes when it is recognised by the world.

Miguel Noguera

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
1.1 Aims	1
1.2 Approach and Context	2
1.3 Contributions	3
1.4 Outline	5
1.5 Publications	6
2 Spiking Neural Networks	8
2.1 Biological Neurons	8
2.2 Spiking Neuron Models	11
2.2.1 Overview of models	12
2.2.2 Leaky Integrate and Fire model (LIF)	15

2.3	Spiking Neural Networks	19
2.3.1	Synaptic models	20
2.3.2	Populations of neurons	22
2.3.3	Neurons and noise	25
2.4	Training Spiking Neural Networks	28
2.4.1	Spiking Networks as Universal Function Approximators	29
2.4.2	Unsupervised Learning	31
2.4.3	Supervised Learning	34
2.4.4	Reinforcement Learning	40
3	Neural Heterogeneity Promotes Robust Learning	43
3.1	Introduction	43
3.2	Results	46
3.2.1	Time scale heterogeneity improves learning on tasks with rich temporal structure	46
3.2.2	Predicted time constant distributions match experimental data	52
3.2.3	Generalisation across time scales	52
3.2.4	Robustness against mistuned learning	53
3.3	Methods	58
3.3.1	Neuron and synaptic models	58
3.3.2	Neural and synaptic model discretisation	59

3.3.3	FORCE Training	64
3.4	Discussion and conclusions	67
4	Sparse Spiking Gradient Descent	70
4.1	Introduction	70
4.2	Methods	73
4.2.1	Forward Model	73
4.2.2	Backward Model	75
4.2.3	Visualisation of $\delta_j[t]$ computation	79
4.2.4	Complexity analysis	80
4.2.5	Sparse learning rule interpretation	82
4.3	Results	83
4.3.1	Spiking neurons are inactive most of the time resulting in higher energy efficiency	84
4.3.2	Sparse backpropagation training approximates the gradient very well	85
4.3.3	Sparse training is faster and less memory intensive	86
4.3.4	Training deeper and more complex architectures	89
4.3.5	Sparsity-Accuracy trade-off	90
4.4	Discussion and Conclusions	91

5 Spiking Network Initialisation and Firing Rate Collapse	94
5.1 Introduction	94
5.2 Background Theory	96
5.2.1 Initialisation based on variance flow in ANNs	97
5.2.2 Population Firing rate in Spiking Neural Networks	102
5.2.3 Threshold integration	110
5.3 Firing Rate Collapse	113
5.3.1 Firing Rate Collapse problem identification	114
5.3.2 Hypothesis	116
5.3.3 General assumptions and proposed solutions	117
5.3.4 Input spikes as a random walk	119
5.3.5 Input spikes as a Wiener process	124
5.3.6 General case	129
5.4 Vanishing-Exploding gradients on SNNs	131
5.4.1 Variance propagation in the forward pass of an SNN	131
5.4.2 Variance propagation in the backward pass of an SNN	133
5.4.3 Computing $\rho_a^{(l)}$	135
5.5 General SNN Initialisation strategy	138
5.6 Discussion and Conclusions	141

6	Conclusion	145
6.1	Future Work	145
6.2	Summary of Thesis Achievements	148
	Appendices	151
A	Gradient of an SNN layer	152
B	Supplementary Information on Sparse Spiking gradient descent	154
B.1	Training details	154
B.1.1	Conversion of Fashion-MNIST to spike latencies	154
B.1.2	Spiking neuron model	154
B.1.3	Network and weight initialisation	156
B.1.4	Supervised and regularisation losses	156
B.1.5	Surrogate gradient	157
B.1.6	Parameters	158
B.2	Loss evolution for all datasets	159
C	Mean and variance of Poisson spikes filtered by the membrane	160
D	Threshold Integration in the shot-noise case	166
E	Supplementary results on Firing rate collapse correction	170
F	Proofs on Vanishing-Exploding gradients on SNNs	171
G	Network Initialisation limit cases	177
	Bibliography	178

Chapter 1

Introduction

1.1 Aims

The main aim of this thesis is to advance the state of the art of deep spiking neural networks (SNNs) training. In particular, we contribute to developing new methods for training SNNs that give more accurate results, are more robust, faster and more memory-efficient, while also improving our understanding of the brain.

Accuracy. Accuracy may be defined as the degree to which an SNN is capable to solve the task it was trained for. Although, artificial neural networks (ANNs) outperform SNNs in terms of accuracy in almost all benchmarks, the difference in performance has steadily decreased in recent years. In this thesis, we aim to achieve better accuracy than previous works on SNNs in an attempt to bridge the gap between ANNs and SNNs.

Robustness. Training SNNs has proven to be more difficult than training ANNs due to a combination of factors such as the spiking non-linearity, hyperparameter tuning and parameter initialisation. This leads to spending more time manually tuning SNNs to achieve effective training. Easing this process is fundamental for

the development of SNN training. We tackle some of these issues in Chapters 3 and 5.

Energy efficiency. One of the main reasons why SNNs are an attractive alternative to ANNs is their low energy consumption. This is usually due to two factors: sparse weights and sparse activations. While sparse weights are applicable to ANNs and have been widely studied in the ANN literature, sparse activations are a consequence of the binary nature of the signals spiking neurons use to communicate. Understanding and harnessing how can learning take place under this condition is fundamental to be able to develop and implement energy efficient SNNs algorithms both for inference and training. We tackle the issue of efficient SNN training in Chapter 4.

Better understanding of the brain. Spiking neurons models were originally developed with the aim of modelling biological neurons. They better represent the dynamical nature and biophysical processes of neurons than less biologically accurate models such as ANNs. Thus, their study is inevitably linked to the study of computation on the nervous system and by studying SNNs we additionally aim to better understand the brain. Some of the results we obtain in Chapter 3 provide evidence to support hypothesis on the functional role of certain characteristics of the brain.

1.2 Approach and Context

In this thesis, deep learning methods are combined with computational neuroscience theory and models. The field of computational neuroscience deals with the study and modelling of the nervous tissue and aims to understand, describe and predict its computational capabilities. This takes place at several levels of abstraction, from the study of complex biophysical processes taking place in cells to the study of large

brain regions.

In recent years, the field of machine learning has experienced a revolution thanks to the combination of deep learning architectures, large datasets and the availability of powerful parallel processing hardware and algorithms. This has resulted in solving, for the first time, highly complex problems in visual and auditory tasks [1, 2, 3], natural language processing [4, 5], games [6, 7, 8] and even solving long standing scientific problems like protein folding [9].

In this context, it is only natural for deep learning techniques to be used and adapted to computational neuroscience models. In particular, the study of spiking neural networks is specially well suited to benefit from deep learning techniques. On one hand, they are simple and similar enough to be simulated efficiently with the same hardware and libraries used for ANNs. On the other hand, SNNs are believed to be a good model for the brain as they are able to reproduce several of its properties as well as reproduce its electrical behaviour fairly accurately. Moreover, there is evidence to suggest that SNNs should be able to perform similarly or better than ANNs while consuming a fraction of the power. This has lead to very promising efforts to bridge these fields [10, 11, 12] suggesting that SNNs combined with deep learning techniques will potentially allow to improve the energy efficiency of intelligent systems and further our understanding into how computation takes place in the brain.

1.3 Contributions

This thesis aims to develop new methods for faster, more accurate, more efficient and more robust SNN training. The contributions of this work can be summarised as follows:

1. **Functional role of heterogeneity of time constants in SNNs.** We

trained spiking neural networks at difficult classification tasks under different heterogeneity schemes. Our main findings were that heterogeneity improved the overall performance across a range of tasks and training methods, specially on tasks with richer temporal structure. The time constants found after training qualitatively matched time constants measured in experimental data. Since the cost of adding heterogeneity of time constants scales with the number of neurons ($O(n)$) and not with the number of synapses ($O(n^2)$) we conclude that neural heterogeneity is a computationally efficient strategy. Moreover, since heterogeneity has no metabolic cost for the brain it supports the hypothesis that it has a functional role and is not simply a product of noisy developmental processes.

2. **Role of heterogeneity against mistuned hyperparameters.** Heterogeneous networks were equally or more robust to mistuned hyperparameters. Learning converges not only to better accuracies but also faster and it is less likely to get stuck in local minima.
3. **Time, energy and memory efficient training algorithm on SNNs.** We introduce a novel training algorithm which allows to backpropagate the gradient with sparse operations on SNNs and developed custom CUDA kernels to show its effectiveness in practice. It achieves the same performance as standard dense backpropagation at a much reduced computational and memory cost (up to 150x faster and 85% more memory efficient). To the best of our knowledge this is the first backpropagation method that leverages the sparse nature of spikes in training and tackles the main performance bottleneck in neuromorphic training.
4. **Theoretical and practical solution to firing rate collapse.** We identify an important problem regarding the accuracy of diffusion and shot-noise models for predicting the firing rate at typical simulation timestep lengths. We

provide a theoretical explanation and solutions to this problem under different assumptions and show that this problem can be easily corrected.

5. **Theoretical and empirical results for optimal weight initialisation on SNNs.** We find a bound on the output firing rate on an L -layer SNNs and obtained a close-form expression to ensure that the SNN gradients do not explode or vanish with the number of layers. We also develop a method to practically calculate the optimal weight initialisation and surrogate function without the need to simulate the network.
6. **Review of methods for computing the firing rate response of LIF networks.** We review and explain step by step the main methods for computing the firing rate and membrane potential distribution of LIF networks.

1.4 Outline

The outline of the thesis is as follows:

Chapter 2: Background on Spiking Neural Networks. In this chapter we give an overview of different types of neuron models according to their level of detail. We describe the leaky integrate and fire model as well as several synaptic models, populations of neurons and the effect of noise in the population activity. Finally we review different SNN training methods including unsupervised, supervised and reinforcement learning methods.

Chapter 3: Neural Heterogeneity promotes robust learning. This chapter presents our work on neural heterogeneity where we study the performance and robustness of recurrent SNNs under different heterogeneity schemes on two different supervised learning methods.

Chapter 4: Sparse Spiking Gradient Descent. This chapter introduces a highly efficient training method based on extending the sparsity present in the forward pass on an SNN (spikes) to the backward pass (gradients). This leads to up to 150x faster and 85% more memory efficient SNN training.

Chapter 5: Optimal Spiking Network Initialisation. In this chapter we first introduce some background theory on optimal weight initialisation on ANNs as well as different methods to compute the firing rate and membrane distribution of a population of spiking neurons. We next identify and solve the firing rate collapse problem for large simulation time steps. Finally, we provide theoretical and empirical results on the vanishing/exploding gradient problems on SNNs and a strategy to optimally initialise the weights and surrogate function on an SNN.

Chapter 6: Conclusion. This last chapter summarises the main contributions and results of this thesis and introduces potential future research directions.

1.5 Publications

The publications derived from the work in this thesis are

- Nicolas Perez-Nieves, and Dan Goodman. Sparse spiking gradient descent. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021
- Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Neural heterogeneity promotes robust learning. *Nature communications*, 2021
- Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Advantages of heterogeneity of parameters in spiking neural network training. *Cognitive Computational Neuroscience (CCN)*, 2019

In addition to the above publications, I have also been involved in other works during my Ph.D., which are not discussed in this thesis:

- Nicolas Perez-Nieves and Dan Goodman. Generalisation of frequency mixing and temporal interference phenomena through Volterra analysis *Computational Neuroscience Conference (CNS)*, 2019
- Nicolas Perez-Nieves, Yaodong Yang, Oliver Slumbers, David H. Mguni, Ying Wen, and Jun Wang. Modelling behavioural diversity for learning in open-ended games. *International Conference on Machine Learning (ICML)*, 2021
- Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *Conference on Robotic Learning (CoRL)*, 2020
- David H. Mguni, Jianhong Wang, Taher Jafferjee, Nicolas Perez-Nieves, Wenbin Song, et al. Learning to Shape Rewards using a Game of Switching Controls. *AAAI Conference on Artificial Intelligence (AAAI)*, 2023
- David H. Mguni, Taher Jafferjee, Jianhong Wang, Nicolas Perez-Nieves, Oliver Slumbers, Feifei Tong, Yang Li, et al. LIGS: Learnable Intrinsic-Reward Generation Selection for Multi-Agent Learning. *International Conference on Learning Representations (ICLR)*, 2022
- Le Cong Dinh, Yaodong Yang, Zheng Tian, Nicolas Perez Nieves, Oliver Slumbers, David Henry Mguni, Haitham Bou Ammar and Jun Wang. Online Double Oracle. *Transactions on Machine Learning Research*, 2022.
- Dániel L. Barabási, Taliesin Beynon, Ádám Katona and Nicolas Perez-Nieves. Complex Computation from Developmental Priors. *Nature communications*, 2023
- Adaptive Agents Team. Human-Timescale Adaptation in an Open-Ended Task Space. *International Conference on Machine Learning (ICML)*, 2023

Chapter 2

Spiking Neural Networks

This chapter presents an overview of spiking neurons. This includes the biological and theoretical foundations of spiking neuron models, spiking neural networks and different techniques developed to train them.

2.1 Biological Neurons

Neurons were first proposed as the basic unit of the nervous system by Wilhelm von Waldeyer-Hartz in the late 19th century and were first observed under the microscope by Santiago Ramón y Cajal between 1897 and 1904 [13]. Since then, the brain has been studied across different scales and levels of abstraction resulting in a vast amount of theoretical and empirical knowledge about its structure and function. Most of this work revolves around the notion that neurons are its elementary processing units (although recent work suggests that glial cells in the nervous system may also play a computational role [14, 15, 16]). The human brain is a remarkable structure containing an order of 10^{11} neurons and an estimated 10^{14} to 10^{15} synapses [17, 18] while only consuming roughly 20W of power [19]. The study of the brain has usually followed either a top-down or a bottom-up approach. The top-down

approach usually starts from an overall behaviour and relates it to the interactions between brain regions. This includes much of the work done in psychology and neuropsychology. The bottom-up approach starts from the interaction between individual neurons (or even more fundamental units) and attempts to explain neural and network properties. This includes disciplines such as molecular and cell biology, biophysics and computational neuroscience [20].

Most work on this thesis aligns with the bottom-up approach and uses neuron models from computational neuroscience. These models attempt to mathematically describe the physiology and empirical observations made on animal neurons. Biological neurons are cells and as such present all standard elements of animal cells (nucleus, cytoplasm, organelles and membrane). However, to study neurons from a computational point of view it is often more useful to divide them into three parts: dendrites, soma and axon as depicted in Fig. 2.1. The main operation of neurons can be roughly summarised as: neurons receive input signals at their dendrites which are then transmitted to the soma, the soma integrates these signals and if it reaches a certain threshold it generates an output signal which is delivered through the axon to other neurons.

Like other cells, neurons contain a wide range of ions and charged molecules. Most of the time there is an excess of negative charge inside neurons resulting in an overall negative *membrane potential* (measured as the voltage difference between the inside and outside of the cell). There are numerous ion channels embedded in the membrane which determine how easy it is for ions to flow in and out the neuron. The conducting capacity of the channels can be modified by many factors including the value of the membrane potential and the external concentration of neurotransmitters. When the net flow of ions is zero the neuron is said to be at rest and the membrane potential remains constant at a *resting potential*. A change in the ion flow balance has the effect of moving the membrane potential away from the

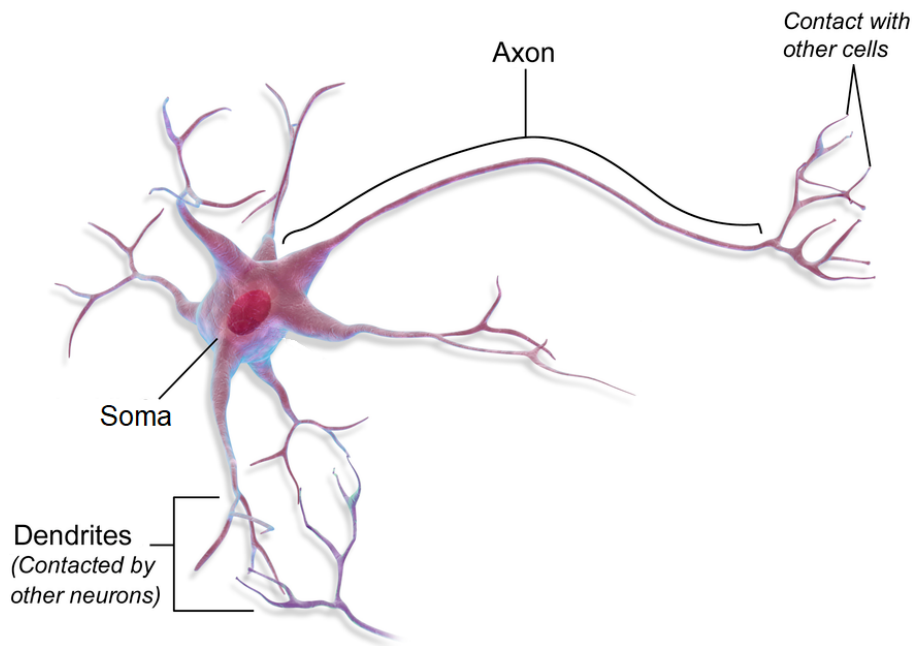


Figure 2.1: Diagram of a biological neuron. Adapted from [21][22] and used under CC BY 3.0 license.

resting potential either in the positive (depolarisation) or negative (repolarisation) direction [23].

Upon reaching a certain *threshold* the neuron emits an action potential from the soma through the axon called a *spike*. The shape of this action potential is usually very stereotypical and as a result most of the literature considers that no information is sent in the shape of the action potential but rather in the fact that an action potential was sent at a given time [24]. This is illustrated in Fig. 2.2 which depicts several membrane potential recordings showing little variability across trials.

Neurons communicate by sending these spikes to other neurons. The junction between two neurons is called a *synapse*. Synapses are unidirectional (from an axon to a dendrite) and we refer to the emitting neuron as *presynaptic neuron* and the receiving neuron as *postsynaptic neuron*. Most synapses are chemical synapses meaning that while the axon terminal is very close to the postsynaptic neuron, there is a small gap between them called synaptic cleft where the presynaptic neuron releases neurotransmitters. More specifically, when an action potential arrives at the synapse

a complex bio-chemical process is triggered which leads to a release of neurotransmitters which are detected at the postsynaptic neuron leading to the opening of specific ion channels and thus changing the net ion flow at the postsynaptic neuron [23, 24].

Despite this seemingly simple operation, biological neurons are highly complex cells that display a wide variety of behaviours. This has led to neural models of varying degrees of abstraction and complexity ranging from simplified models consisting of a single equation to biologically accurate models consisting of thousands of coupled differential equations. Highly detailed models attempt to reproduce a long list of properties and phenomena such neural and dendritic morphology, synaptic adaptation, bursting, postinhibitory rebounds etc. [23, 24]. It is not clear which properties are necessary to create a network capable of displaying intelligent behaviour and which ones are not. This is further discussed in Chapter 3. Often, for technical reasons, we use simpler models that capture as much as possible of the underlying biological mechanisms but are simple enough to simulate using today's computational resources. This is further discussed in Chapter 4.



Figure 2.2: Membrane potential recordings at four different TRN neurons. Note how the spike shape is highly stereotypical. Adapted from [25][26] and used under CC BY-SA 4.0 license.

2.2 Spiking Neuron Models

Traditionally, the end goal of neuron models has been to explain the mechanisms behind the operation of the nervous system. However, in recent years there has been

an increasing interest in using spiking neurons for deep learning with the aim of leveraging their unique properties. In this section we present an overview of models and a more detailed description of the Leaky Integrate and Fire model (LIF) which will be the basic model for the rest of this thesis.

2.2.1 Overview of models

Spiking neuron models attempt to describe the relationship between the different currents across the membrane and the membrane potential as well as the generation of spikes. This can be done at different levels of detail that we briefly review here.

Detailed Neuron Models

These models attempt to account for various ion channels, different kinds of synapses and the neuron topology. The Hodgkin-Huxley model [27] was the first model attempting to describe the interplay of various ionic currents measured in a series of experiments on the giant axon of the squid. This model is summarised in Fig. 2.3A. The neuron membrane consists of a thin bilayer of lipids that acts as a capacitor. The ionic pumps and channels in the membrane allow ions to flow in and out the membrane. We distinguish two types of ionic currents corresponding to sodium Na^+ and potassium K^+ as well as a leakage current that accounts for all other ions that are not explicitly modelled. Since the ionic concentrations inside and outside the neuron are different they induce a voltage difference. These are modelled as voltage sources whose polarity depends on the relative ionic concentrations. Since ion pumps actively transport ions from one side of the membrane to the other (consuming energy in the process) the relative ion concentrations result in a non-zero equilibrium potential. Finally, the conductance of the ion channels is voltage and time dependent while the leakage channel is voltage independent.

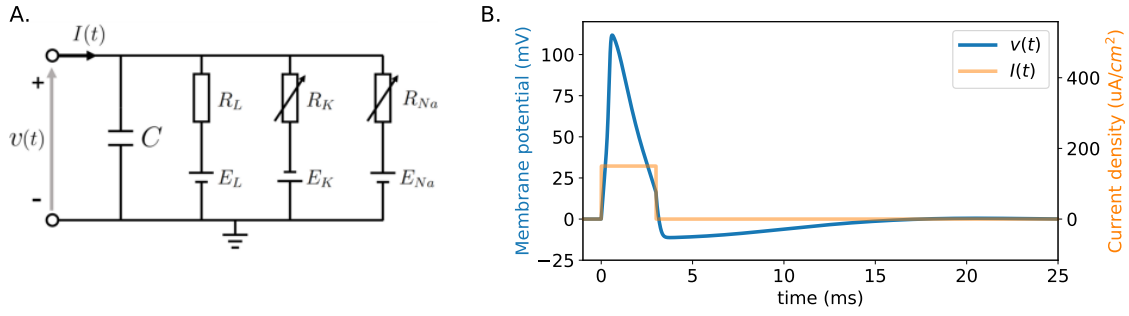


Figure 2.3: Hodgkin-Huxley model. **A.** Circuit diagram equivalent of the Hodgkin-Huxley model. **B.** Simulation of the Hodgkin-Huxley model where a pulsed current generates an action potential (spike).

The full Hodgkin-Huxley model is given by the following four differential equations

$$\begin{aligned}
 C \frac{dv}{dt} &= I(t) - g_{Na} m^3 h (v - E_{Na}) - g_K n^4 (v - E_K) - g_L m^3 h (v - E_L) \\
 \frac{dm}{dt} &= \alpha_m(v)(1 - m) - \beta_m(v)m \\
 \frac{dn}{dt} &= \alpha_n(v)(1 - n) - \beta_n(v)n \\
 \frac{dh}{dt} &= \alpha_h(v)(1 - h) - \beta_h(v)h
 \end{aligned} \tag{2.1}$$

Where the first equation simply applies Kirchhoff current law to the circuit on Fig. 2.3A. The conductances (g_{Na} , g_K , g_L), reversal potentials (E_{Na} , E_K , E_L) and $\alpha(\cdot)$ and $\beta(\cdot)$ functions are experimentally derived. If enough input current $I(t)$ is injected this model the membrane potential will display a spiking profile as that shown in 2.3B. This model is capable of displaying several spiking regimes such as a single spike, periodic spiking and no spiking.

More detailed models have been developed to tackle the limitations of the Hodgkin-Huxley model. These include adding other voltage-activated ion channels [28][29], calcium-activated channels [30] and transmitter activated ion channels to account for synaptic inputs (see [31] Chapter 2 for a more detailed review of ion channels).

Stochasticity can also be added into these models to account for noise in ion channels [32]. Finally, we can also consider the spatial structure of the neuron in our models leading to distinguishing between single-compartment models (also called point-neuron models) such as the Hodgkin Huxley model and multi-compartment models where the neuron is represented as a collection of discrete compartments (see [31] Chapter 2.6).

Phenomenological spiking models

Detailed models while more accurate from a physiological point of view are difficult to analyse and computationally expensive to simulate. For this reason simpler models have been developed that attempt to model the neuronal dynamics while not having to resort to specific biophysical details. Some of the most popular are two-dimensional neuron models as reducing the number of coupled differential equations to two allows to easily study the neuronal dynamics in a phase plane. These equations often describe how the membrane potential and another variable with no biophysical meaning vary over time. Models like this include the FitzHugh-Nagumo [33][34], Morris-Lecar [35] and Hindmarsh–Rose [36] among others (see [31] Chapter 3 for more).

Two-dimensional neuron models can be yet simplified further by considering that spikes are highly stereotypical events whose shape does not contain any specific information [24]. This leads to even simpler models consisting of a single equation (often a differential equation) followed by a non-linearity (usually the Heaviside step function) that decides when the spiking mechanism takes place. The differential equation governs the behaviour of the membrane potential in the *subthreshold regime* then if the membrane potential reaches a certain threshold, the neuron emits a spike and resets to the resting potential. Note that in these models spikes are formal events that occur at particular points in time. Thus, the membrane potential of this model

will not reflect the characteristic spike shape but simply a discontinuity due to the resetting. This is shown in Fig. 2.4A.

These models, often called formal spiking models, are very popular for the study of neural coding, memory and larger network dynamics as they are computationally inexpensive. Some of the most widely used models include the integrate-and-fire [37], leaky integrate-and-fire [37], exponential integrate-and-fire [38], spike response model [39][40] and fractional-order leaky integrate and fire model [41] among many others (see [31] Chapter 4 for more).

While these models are attractive due to their simplicity they show limitations regarding their ability to reproduce the behaviour of more detailed models. These differences can be improved by adding other mechanisms such as threshold adaptation and refractory periods leading to models where spikes are still formal events but are more biologically realistic.

2.2.2 Leaky Integrate and Fire model (LIF)

The leaky integrate-and-fire (LIF) model was first introduced by Louis Lapicque in 1907 [37] and over a hundred years later it is still one of the most popular models due to its simplicity. We will explain this model in more detail as it will be the basic neuron model used in the following chapters.

In this model the membrane potential $v(t)$ varies over time according to

$$\tau_m \frac{dv(t)}{dt} = -(v(t) - V_0) + RI(t), \quad \text{if } v(t) < V_{th} \quad (2.2)$$

The LIF model can be interpreted as a simplification of the Hodgkin-Huxley model in which all currents have been lumped together into the input current $I(t)$, V_0 is the

resting potential and $\tau_m = RC$ where C is the membrane capacitance and R is the membrane resistance. Upon reaching the threshold V_{th} three things happen: a spike is emitted, the membrane is reset to V_r and the neuron enters a refractory period that lasts t_{ref} seconds where the neuron cannot spike and the membrane potential is kept constant at V_r .

Spikes are now formal events which means that the output signal of the neuron (the signal that is to be transmitted though the synapse) can be written as a sum of delta functions taking place at the spike times $\{t_{(f)}\}$. This stream of spikes is called a *spike train* and can be formalised as

$$s(t) = \sum_{t^{(f)}} \delta(t - t^{(f)}) \quad (2.3)$$

To give an intuition of the LIF neuron operation and its parameters we plot several neuron traces in Fig. 2.4. Fig. 2.4A shows two neurons with different time constants $\tau_m = 10, 20\text{ms}$ respectively and sharing the other parameters $V_0 = V_r = -70\text{mV}$, $V_{th} = 10\text{mV}$, $t_{ref} = 0$, $R = 1\Omega$. Both of them receive a constant input current $I(t) = 90\text{mA}$. Note that there is no ‘spike shape’ in the traces at all. As mentioned before, the spikes are now formal events that we can record and transmit to other neurons as we will see later but are not part of the membrane equation. It is not uncommon to manually add the spikes just for plotting purposes as shown in Fig. 2.4B. We will use this kind of depiction in the following figures.

The LIF operation is simple, the input current $I(t)$ is integrated resulting in an increase of the membrane potential until it reaches the threshold. The speed at which the membrane potential increases is determined by the time constant τ_m . A longer time constant results in a lower charging speed. After reaching the threshold, the neuron resets to V_r .

Figure 2.4C shows the same trace but now the input current becomes 0mA after 50ms

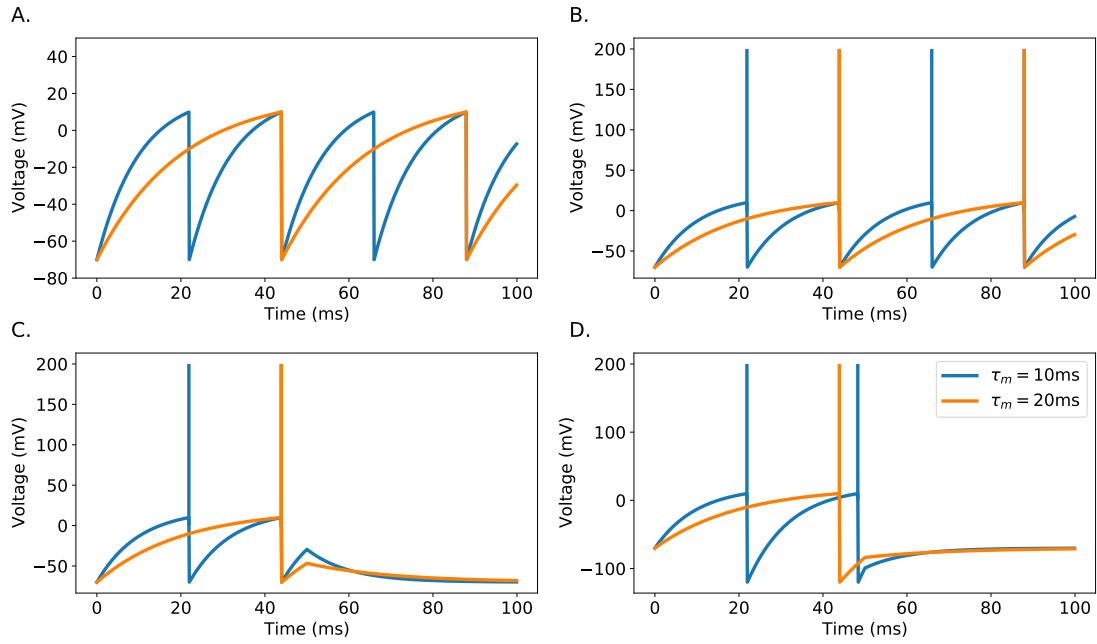


Figure 2.4: Membrane potential of two LIF neurons with different time constants. We add the ‘spike shape’ artificially at spike times for visualisation. **A.** Membrane potential under a 90mA constant input current where the ‘spike shape’ has *not* been added. **B.** Membrane potential under a 90mA constant input current. **C.** Membrane potential under a pulsed current of 90mA between 0 and 50ms and 0mA afterwards. **D.** Membrane potential under a pulsed current of 90mA between 0 and 50ms and 0mA afterwards with a resting potential ($V_0 = -70\text{mV}$) different from the reset potential ($V_r = -120\text{mV}$).

resulting in the membrane potential turning towards the resting potential. Looking back into equation (2.4), we observe that the resting potential and a constant input current are identical from a mathematical point of view. In this case, we can write the right hand side as $-v(t) + V_0 + RI = -v(t) + \hat{V}_0 = -v(t) + R\hat{I}$. This means that we can interpret the resting potential V_0 simply as a constant current into the neuron or alternatively, a constant current into the neuron as a resting potential. In the following sections we will use the following equation for the LIF model instead and assume that if a rest potential needs to be added it can be simply be part of the input current $I(t)$.

$$\tau_m \frac{dv(t)}{dt} = -v(t) + RI(t), \quad \text{if } v(t) < V_{th} \quad (2.4)$$

Finally, note that in all examples the resting potential and the reset potential are the same. This is common practice in the literature but it is not necessarily always the case. An example of this is show in 2.4D.

Subthreshold regime exact solution

Equation (2.4) is nothing but a linear first order differential equation and thus we can obtain an exact solution. Provided an initial condition $v(t_0) = v_0$ we can write the solution to equation (2.4) for $t \geq t_0$ in several equivalent ways:

$$\begin{aligned}
 v(t) &= v_0 \exp\left(-\frac{t-t_0}{\tau_m}\right) + \frac{R}{\tau_m} \exp\left(-\frac{t}{\tau_m}\right) \int_{t_0}^t \exp\left(\frac{s}{\tau_m}\right) I(s) ds \quad (2.5) \\
 &= v_0 \exp\left(-\frac{t-t_0}{\tau_m}\right) + \frac{R}{\tau_m} \int_0^{t-t_0} \exp\left(-\frac{s}{\tau_m}\right) I(t-s) ds \\
 &= v_0 \epsilon(t-t_0) + \frac{R}{\tau_m} (\epsilon * I)(t-t_0)
 \end{aligned}$$

where $\epsilon(s) = \exp(-s/\tau_m)H(s)$, $H(t)$ is the Heaviside step function and $*$ is the convolution operation. Looking at equation (2.5) we note that the membrane is simply filtering the input current and the filter response is a decaying exponential. Note however that this solution is only valid as long as $v(t)$ does not cross the threshold. This is what we often call the *subthreshold regime*. After crossing the threshold, we would need to reset the value of $v(t)$ and then continue integrating.

To simulate a LIF neuron on a computer we need to discretise time. Knowing that at a given t_0 the membrane potential is $v(t_0) = v_0$ we want to compute what the membrane potential will be after a short time interval of length Δt . We assume that the current is constant during the interval $[t_0, t_0 + \Delta t)$ (see Chapter 5 for the implications of this assumption). Thus, equation (2.4) can be rewritten as

$$\begin{aligned}
v(t_0 + \Delta t) &= v(t_0) \exp\left(-\frac{\Delta t}{\tau_m}\right) + \frac{RI(t_0)}{\tau_m} \exp\left(-\frac{t}{\tau_m}\right) \int_{t_0}^t \exp\left(\frac{s}{\tau_m}\right) ds \quad (2.6) \\
&= v(t_0) \exp\left(-\frac{\Delta t}{\tau_m}\right) + RI(t_0) \left(1 - \exp\left(-\frac{\Delta t}{\tau_m}\right)\right) \\
&= \alpha v(t_0) + (1 - \alpha) RI(t_0)
\end{aligned}$$

We will often also write the equations as a recurrent relation

$$v[n + 1] = \alpha v[n] + (1 - \alpha) RI[n] \quad (2.7)$$

with $n \in \{0, 1, \dots\}$ and $v[0] = v_0$.

We can also obtain the spikes in discrete time by simply applying

$$s[n + 1] = H(v[n + 1] - V_{th}) \quad (2.8)$$

where $H(x)$ is the Heaviside step function.

2.3 Spiking Neural Networks

The brain is an enormous network of interconnected neurons hierarchically organised according to different areas, subregions and layers. As such, understanding the brain requires studying how different neurons interact and communicate with each other. In this chapter we will visit different types of synaptic models, populations of neurons, how to simulate spiking neural networks (SNNs) in discrete time and finally how to model the presence of noise in these networks.

2.3.1 Synaptic models

Synapses establish how neurons communicate to each other. After a spike has been emitted it is propagated through the axon until it meets with other neurons through a connection called a synapse. The process of transmitting the spiking signal to the next neuron is complex as it involves the release of neurotransmitters into the synaptic cleft, the opening of channels in the postsynaptic neuron and the speed at which the presynaptic neuron recovers and is able to release neurotransmitter again. We often use much simplified synaptic models that simply inject a current that is a low-pass-filtered version of the presynaptic spike.

The simplest model is a simple first order filter which we can model as a first order linear equation

$$\tau_s \frac{I(t)}{dt} = -I(t) + \sum_{t^{(f)}} w \delta(t - t^{(f)}) \quad (2.9)$$

where we have introduced a synaptic time constant $\tau_s > 0$ and a weight $w \in \mathbb{R}$ which simply scales the synaptic response. Synapses can be *excitatory* or *inhibitory* depending on the weight w being positive or negative respectively. The solution to this equation is simply the convolution of a sum of deltas with the kernel $\epsilon(s) = \exp(-s/\tau_s)H(s)$ leading to

$$I(t) = \sum_{t^{(f)}} w \exp\left(-\frac{t - t^{(f)}}{\tau_s}\right) H(t - t^{(f)}) \quad (2.10)$$

given an initial condition $I[t_0] = 0$ and $t^{(f)} > t_0 \forall t^{(f)}$. Fig. 2.5B shows a synaptic current resulting from this kind of filtering.

More complex synapses are also often used consisting of a raising and a decaying time constant $\epsilon(s) = (1 - \exp(-s/\tau_r)) \exp(-s/\tau_s)H(s)$ leading to synaptic currents

$$I(t) = \sum_{t^{(f)}} w \left(1 - \exp \left(-\frac{t - t^{(f)}}{\tau_r} \right) \right) \exp \left(-\frac{t - t^{(f)}}{\tau_s} \right) H(t - t^{(f)}) \quad (2.11)$$

Fig. 2.5C shows the resulting synaptic current.

In addition, some synaptic models also include a fast and a slow decaying rates $\epsilon(s) = [1 - \exp(-s/\tau_r)][\exp(-s/\tau_{fast}) + \exp(-s/\tau_{slow})]H(s)$ leading to

$$I(t) = \sum_{t^{(f)}} w \left[1 - \exp \left(-\frac{t - t^{(f)}}{\tau_r} \right) \right] \left[\exp \left(-\frac{t - t^{(f)}}{\tau_{fast}} \right) + \exp \left(-\frac{t - t^{(f)}}{\tau_{slow}} \right) \right] H(t - t^{(f)}) \quad (2.12)$$

shown in Fig. 2.5D. Note how the decaying time is slower now compared to Fig. 2.5C.

In general, we can have any causal kernel convolved with the sum of spikes leading to synaptic input currents of the form

$$\begin{aligned} I(t) &= \sum_{t^{(f)}} w \int_{t_0}^t \epsilon(s) \delta(t - t^{(f)}) \\ &= \sum_{t^{(f)}} w \epsilon(t - t^{(f)}) \end{aligned} \quad (2.13)$$

Finally, if we consider the synaptic time constant(s) to be very small relative to the membrane time constant we can neglect the membrane kernel altogether and end up with *delta* or *instantaneous* synapses. This model is often used in the literature to simplify theoretical analysis and gain computational speed and will be used in the following chapters. In this model, synaptic filtering is completely bypassed and spikes (a sum of delta functions) are directly integrated into the membrane giving a total synaptic current

$$I(t) = \sum_{t^{(f)}} w \delta(t - t^{(f)}) \quad (2.14)$$

This kind of input would look like the unfiltered spike train in Fig. 2.5A.

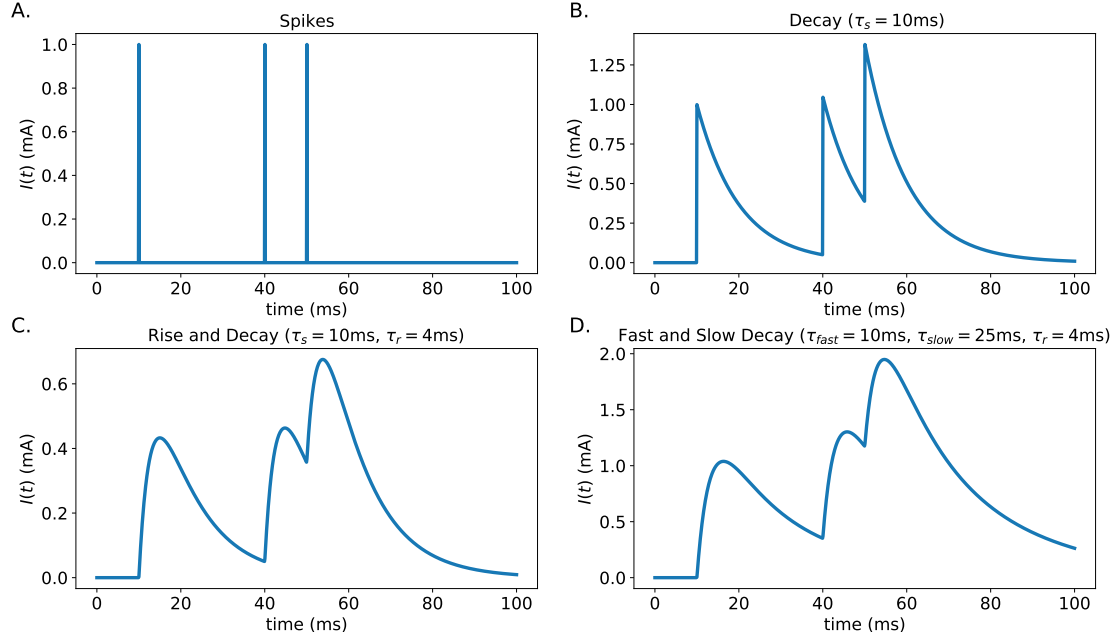


Figure 2.5: Different types of synaptic filtering. We used weight $w = 1$ and the time constants as specified in each plot title. **A.** No filtering, the spikes are directly integrated. **B.** Simple decaying synapse **C.** Rise and decay synapse **D.** Rise and decay synapse with a fast and slow decaying time constant.

2.3.2 Populations of neurons

Neurons in the brain are connected with complex connectivity schemes that are not yet fully known. In this thesis however, we will adopt the typical connectivity schemes present in the machine learning literature where neurons are organised in different layers. Input signals are presented to the first layer, then the output spikes from this layer are the input to the next one and so on. Neurons within a layer may or may not be connected to other neurons in the same layer.

We can write the membrane potential of a layer of LIF neurons receiving inputs from a previous layer with delta synapses (spikes are directly integrated) as

$$\tau_m \frac{dv_j^{(l)}(t)}{dt} = -v_j^{(l)}(t) + RI_j^{(l)}(t) + \sum_i w_{ij}^{(l-1)} s_i^{(l-1)}(t), \quad \text{if } v_j^{(l)}(t) < V_{th} \quad (2.15)$$

where l is the layer index, i is the presynaptic neuron index, j is the postsynaptic neuron index, $w_{ij}^{(l-1)}$ is the weight from neuron j in layer $l-1$ to neuron i in layer l . The incoming spike trains $s_i^{(l-1)}(t)$ is now defined similarly to (2.3) as

$$s_i^{(l-1)}(t) = \sum_{t_i^{(f)}} \delta(t - t_i^{(f)}) \quad (2.16)$$

where we have omitted the superscript l in the times $t_i^{(f)}$ for notational simplicity.

Note that we are using homogeneous values for the threshold, resistance and time constant (i.e. all neurons in layer l share the same parameters). This is a common simplification that is often adopted in the literature that will be further studied in Chapter 3.

As before we can compute the solution of (2.15) since it is simply a linear ODE

$$\begin{aligned} v_j^{(l)}(t) = v_0 \exp\left(-\frac{t-t_0}{\tau_m}\right) + \frac{R}{\tau_m} \int_{t_0}^t \exp\left(-\frac{s-t}{\tau_m}\right) I_j^{(l)}(s) ds \\ + \sum_i \sum_{t_i^{(f)}} w_{ij} \exp\left(-\frac{t-t_i^{(f)}}{\tau_m}\right) H(t-t_i^{(f)}) \end{aligned} \quad (2.17)$$

Discretising this solution leads to a problem since now we need to consider what happens with spikes that take place within the interval $(t, t + \Delta t]$. Usually this is dealt with by placing all spikes either at the beginning or at the end of the interval. This problem is analogous to the problem of discretising $I(t)$ by assuming it

constant within the timestep interval, yet it leads to important inaccuracies for some theoretical predictions when Δt is not small enough. This is further investigated in Chapter 5. Leaving all spikes at the beginning of the interval leads to the following expression

$$\begin{aligned} v_j^{(l)}(t_0 + \Delta t) &= v_j^{(l)}(t_0) \exp\left(-\frac{\Delta t}{\tau_m}\right) + RI_j^{(l)}(t_0) \left(1 - \exp\left(-\frac{\Delta t}{\tau_m}\right)\right) + \sum_i \sum_{t_i^{(f)}} w_{ij} \delta(t - t_0) \\ &= \alpha v_j^{(l)}(t_0) + (1 - \alpha) RI_j^{(l)}(t_0) + \sum_i w_{ij} \delta_i \end{aligned} \quad (2.18)$$

where δ_i is the Kronecker delta function which is 1 if the input neuron i spiked within the interval $(t, t + \Delta t]$ and we are assuming that neurons can spike at most once within every interval. As a recurrent equation this gives

$$v_j^{(l)}[t + 1] = \alpha v_j^{(l)}[t] + (1 - \alpha) I_j^{(l)}[t] + \sum_i w_{ij} s_i^{(l-1)}[t] \quad (2.19)$$

where again $s_i^{(l-1)}[t]$ is 1 if the input neuron i spiked within the interval $(t, t + \Delta t]$.

We also obtain the spikes in discrete time by simply applying

$$s_j^{(l)}[n + 1] = H(v_j^{(l)}[n + 1] - V_{th}) \quad (2.20)$$

We will often be interested in quantifying the level of activity of a layer. We can use the *population activity* for this purpose. This is defined as the proportion of neurons at layer l that fire within a given time interval $[t, t + \Delta t]$.

$$\nu^{(l)}(t) = \frac{1}{n^{(l)}} \sum_{i=1}^{n^{(l)}} \sum_{t^{(f)}} \delta(t - t_j^{(f)}) \quad (2.21)$$

where $n^{(l)}$ is the number of neurons at layer l and $t^{(f)} \in [t, t + \Delta t]$. We can approximate the firing rate using discrete time spikes with

$$\nu^{(l)} = \frac{1}{N n^{(l)}} \sum_{i=1}^{n^{(l)}} \sum_{n=1}^N s[n] \quad (2.22)$$

where N is the total number of simulation steps.

2.3.3 Neurons and noise

Recording biological neurons experimentally under the same conditions yields highly variable results from trial to trial. Spike trains are rarely periodic and firing patterns seem random. There are several sources of noise that result in this behaviour. Some of these include thermal noise [42], the finite number of ion channels [43], synaptic transmission failures [44] and random network connectivity which can result in chaotic dynamics [45, 46, 47]. Answering how the brain encodes and processes information under this highly noisy circumstances is still an open question.

Membrane as an Ornstein-Uhlenbeck process

Noise is often modelled by adding a Gaussian noise term to the membrane differential equation. The resulting equation is a stochastic differential equation. For instance in a LIF model (2.4) we end up with

$$\tau_m \frac{dv(t)}{dt} = -v(t) + RI(t) + \xi(t), \quad \text{if } v(t) < V_{th} \quad (2.23)$$

where $\xi(t) \sim \mathcal{N}(0, \sigma^2 \tau_m)$. This type of equation describes an Ornstein-Uhlenbeck process. The solution to this stochastic equation for $t > t_0$ is a Gaussian process with mean and variance given by

$$\mu_v(t) = \mathbb{E}[v(t)] = v_0 \exp\left(-\frac{t-t_0}{\tau_m}\right) + \frac{R}{\tau_m} \int_0^{t-t_0} \exp\left(-\frac{s}{\tau_m}\right) I(t-s) ds \quad (2.24)$$

$$\sigma_v^2(t) = \text{Var}[v(t)] = \frac{\sigma^2}{2} \left[1 - \exp\left(-\frac{2(t-t_0)}{\tau_m}\right)\right] \quad (2.25)$$

Note that the mean value is nothing but the noise-free trajectory as it was obtained in (2.5). Also note that the variance tends to $\sigma^2/2$ as $t \rightarrow \infty$. This solution is only true in the subthreshold regime. As soon as we add a threshold the distribution is no longer Gaussian. This will be further explored in Chapter 5.

We can simulate this process using the Euler-Maruyama method [48] which is an extension of the original Euler method. This leads to

$$v(t + \Delta t) = \left(1 - \frac{\Delta t}{\tau_m}\right) v(t) + \frac{\Delta t}{\tau_m} RI(t) + \sigma \sqrt{\frac{\Delta t}{\tau_m}} n(t) \quad (2.26)$$

with $n(t) \sim \mathcal{N}(0, 1)$.

Alternatively, for LIF neuron, we can use an exact solution which follows from the fact that $v(t)$ is Gaussian [49], leading to

$$v(t + \Delta t) = \exp\left(-\frac{\Delta t}{\tau_m}\right) v(t) + \left(1 - \exp\left(-\frac{\Delta t}{\tau_m}\right)\right) RI(t) + \sigma \sqrt{\frac{1 - \exp(-2\Delta t/\tau_m)}{2}} n(t) \quad (2.27)$$

with $n(t) \sim \mathcal{N}(0, 1)$ again. Both solutions assume that the input current is constant in the interval $[t, t + \Delta t)$ as in 2.6.

Membrane as a Stein's model

In addition to white noise models we can also model noise in the input spikes times. This leads to the same equation as in (2.15) which we rewrite here but now dropping the neuron and layer indices.

$$\begin{aligned} \tau_m \frac{dv(t)}{dt} &= -v(t) + RI(t) + \sum_i w_i \sum_{t_i^{(f)}} \delta(t - t_i^{(f)}), \\ &= -v(t) + RI(t) + \sum_i w_i s_i(t), \quad \text{if } v(t) < V_{th} \end{aligned} \quad (2.28)$$

The key difference with 2.6 is that spike times $t_i^{(f)}$ are no longer deterministic but are drawn from a probability distribution. In computational neuroscience spikes are often generated by a Poisson process. This means that spikes happen with a rate ν but the specific spike times do not matter. Equation (2.28) is then called Stein's model [50, 51].

The expected number of input spikes from neuron i per unit time then is $\langle s_i(t) \rangle = \nu_i$, where we defined $\langle f(t) \rangle = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{T/2} f(t) dt$. It can also be shown that the autocorrelation $\langle s_i(t) s_i(t') \rangle = \nu_i \delta(t - t') + \nu^2$ [24]. Note that the definition of autocorrelation is in the signal processing sense, not in the probabilistic sense [52]. Using these facts we can obtain the steady-state mean and variance of the membrane potential (i.e. the average value and variance of the membrane after running the integration of spikes generated following this process for a long time). See [24] sections 7.5 and 8.2 for more details.

$$\mu_v(t) = \langle v(t) \rangle = \frac{R}{\tau_m} \int_0^{t-t_0} \exp\left(\frac{-s}{\tau_m}\right) I(t-s) ds + \tau_m \sum_i w_i \nu_i \quad (2.29)$$

$$\sigma_v^2 = \langle [v(t) - \langle v(t) \rangle]^2 \rangle = \frac{\tau_m}{2} \sum_i w_i^2 \nu_i \quad (2.30)$$

where we assumed the initial condition $v_0 = 0$ for simplicity. Note that $\mu_v(t)$ is simply the filtered current (deterministic part) plus some new term which is the average input due to the spikes (stochastic part) after they have been integrated into the membrane for a long time. These expressions will be used again in Chapter 5 to show that under certain conditions the Stein's model in (2.28) converges to the Ornstein-Uhlenbeck process.

Simulating this process is as simple as using (2.19) using a Poisson generator to generate the spikes.

2.4 Training Spiking Neural Networks

One of the most remarkable capabilities of the brain is that it is able to *learn* from data in the form of sensory stimulus. The question of learning from data is the object of research of the field machine learning for which it leverages techniques from statistics, computer science, cognitive science and optimisation among others. In recent years methods based on the usage of multi-layer artificial neural networks have lead to unprecedented success matching and even surpassing human-level performance on increasingly difficult auditory and visual recognition problems, natural language processing, games and various real world problem solving. This has led to the adoption of these methods to train spiking neural networks leading to highly effective training algorithms that perform almost on par with artificial neural networks.

It is worth clarifying that by training we mean the modification of synaptic efficacy (i.e. synaptic weights W_{ij}). There is a large amount of evidence, both theoretical and experimental, that support that learning in the brain is fundamentally driven by the modification of synaptic weights. Similarly, most artificial neural network methods ultimately aim to find an optimal set of parameters (weights) to solve a task. In this section we provide an overview of the different methods that have been developed over the years to train spiking neural networks.

2.4.1 Spiking Networks as Universal Function Approximators

Universal approximation theorems are mathematical results that allow us to state that an algorithmically generated class of functions is dense within a given function space of interest. Informally, universal approximation theorems state that neural networks are capable of approximating any continuous function with any required degree of precision. More precisely, neural networks can approximate continuous function with a compact domain and range arbitrarily well with regard to uniform convergence (i.e for a function $f(x)$ we can always find a neural network $\hat{f}(x)$ such that $|f(x) - \hat{f}(x)| < \varepsilon \forall x$ with $\varepsilon > 0$). There are multiple theorems that state this result under different types of assumptions such as the network depth and width, the activation function used and the class of functions that can be approximated.

By neural network we usually mean a composition of functions

$$\hat{f}(x) = f_L \circ f_{L-1} \circ \dots \circ f_1(x) \quad (2.31)$$

Each function composition corresponds to a new layer with $n^{(l)}$ neurons. The function $f_l : \mathbb{K}^{n^{(l-1)}} \rightarrow \mathbb{K}^{n^{(l)}}$ is usually defined as $f_l(x) = \sigma(W^{(l)}x + b^{(l)})$ and often $\mathbb{K}$ is

the real numbers $\mathbb{R}$ or an interval $[a, b]$. The function $\sigma(x)$ is called the activation function and is applied element wise to each neuron. The parameters that are allowed to change to generate the approximation of $f(x)$ are $W^{(l)} \in \mathbb{R}^{n^{(l-1)} \times n^{(l)}}$ and $b^{(l)} \in \mathbb{R}^{n^{(l)}}$. We will often refer to the set of all parameters in a network as θ .

Some of the most notorious results include proving this for continuous sigmoidal functions (functions that tend to 0 at $-\infty$ and to 1 at ∞) given an arbitrary layer width in multilayer perceptrons with as few as one hidden layer [53, 54, 55]. This results was later extended to a wider class of activation functions [56, 57]. Importantly, these theorems do not provide a means to compute how many neurons per layer are required to satisfy a given approximation accuracy.

The arbitrary depth case has also been widely studied in [58, 59, 60, 61, 62, 63] as well as the case for bounded width and depth [64, 65, 66]. Similar results have been obtained for other types of networks such as recurrent neural networks (RNNs) [67] and convolutional neural networks (CNNs) [68].

The case for spiking neural networks has also been studied in [69] and [70]. The main result states that any feedforward or recurrent neural network with s neurons that use saturation function activations (the simplest type of sigmoidal) can be simulated by a network of $s + c$ spiking neurons (where c is a small constant) where the *inputs and outputs are encoded in temporal delays of spikes* [70]. Thus, according to this theorem, for spiking neural networks to be universal function approximators in the same sense as sigmoidal networks it is sufficient to encode the input and output values in the temporal delays of spikes. That is, the output values should be encoded as $t_j = T_{out} - \sigma(\sum_i W_{ij}x_i)$, where $\sigma(x)$ is the saturation function $\sigma(x) = x \cdot \mathbb{1}(0 < x < 1) + \mathbb{1}(x \geq 1)$. More recently, it has also been shown that SNNs with alpha synapses that encode the output in the relative timing of the output spikes are also universal functions approximators [71].

However, as we will see in the following sections and chapters, most recent work using

spiking networks uses a function of the membrane potential at the last layer (e.g. the sum of the membrane potential over time) to compute the output values instead of the spike times as described above. In these models, spiking neural networks are often discretised and simulated in such a way that they are essentially a special case of recurrent neural networks (RNNs) where the activation function is the Heaviside step function. Under these conditions it should be possible to prove that SNNs are universal function approximators however, to the best of our knowledge, this result has only been proved for the saturation function, and thus, there is no formal proof in the literature for the spiking case. Despite this, there is a wide range of empirical results showing that SNNs can successfully learn from data and perform almost on par to other artificial neural networks.

Universal function approximation results are important because they demonstrate that it is *possible* to find a set of parameters which approximate a function within a certain bound. Nevertheless, they do not yield any method or algorithm to obtain this set of parameters in tractable time. The following sections provide an overview of methods developed to deal with this problem in the case of SNNs.

2.4.2 Unsupervised Learning

In unsupervised learning a learning system receives a sequence of input data samples $\{x_i \in \mathbb{R}^n : i = 1, 2, \dots, N\}$ and aims to build useful representations of the data to be used for decision making, predicting future inputs, compressing the inputs etc. It can be thought of as methods that aim to identify statistically salient features in the data [72]. Unsupervised learning methods have been extensively used in the context of spiking neural networks mostly as Hebbian learning rules and variations of spike-timed-dependent plasticity which we briefly review here.

Hebbian learning rules

In 1949 Hebb postulated that when a presynaptic neuron stimulates a postsynaptic neuron repeatedly some metabolic changes lead to making it easier for the presynaptic cell to stimulate the postsynaptic cell [73]. This was later rephrased in a more concise way by Shatz as ‘*Neurons that fire together wire together*’ [74]. This postulate resulted in many experimental work that confirmed to an extent that this is indeed the case in biological neurons as well as many theoretical results and formulations of learning rules.

We can write a general Hebbian learning rule as

$$\frac{dW_{ij}}{dt} = f(W_{ij}, \hat{\nu}_i, \hat{\nu}_j) \quad (2.32)$$

where $\hat{\nu}_i$ and $\hat{\nu}_j$ are some measure of the neural activity of the presynaptic and postsynaptic neurons respectively (not necessarily the firing rate). These rules are local, meaning that the weight update only depends on the neurons connected by the corresponding synapse. The function $f(\cdot)$ should be defined such that a weight change only happens if both neurons are simultaneously active.

The simplest case is the basic Hebb rule [23]:

$$\tau_w \frac{dW_{ij}}{dt} = \nu_i \nu_j \quad (2.33)$$

where ν_i, ν_j are firing rates and $\tau_w > 0$ the learning time constant. This rule however is unstable since the absolute value of the weight can grow indefinitely. This can be solved by a modification such as Oja’s learning rule which is stable [75].

It is worth noting that a Hebbian learning rule is an unsupervised learning method since synaptic changes happen whenever there is joint activity on both neurons regardless of whether this activity is beneficial or not for solving a task. Thus,

Hebbian learning can only learn correlations in activity of both neurons. It has been shown that Hebbian learning rules, if applied for long enough while presenting some input data, result in weight vectors that are parallel to the principal eigenvectors of the correlation matrix of the inputs [75]. In this sense, Hebbian learning rules perform an operation similar to Principal Component Analysis (PCA) which is a classical unsupervised learning method often used for dimensionality reduction [76].

Hebbian learning rules have been successfully used to model the development of ocular dominance [77], orientation selectivity [78, 79], concept acquisition [80] and a computational model of infant cognitive development [81] among others. However, since Hebbian learning is an unsupervised local learning rule it is unable to solve a more general class of problems that require minimising a loss function.

Spike-timing-dependent plasticity (STDP)

While Hebbian learning rules usually concentrate around using the firing rates to change the weights, learning rules based on spike-timing-dependent plasticity (STDP) work at the specific spike time level. In fact, STDP can be seen as a spike-based version of a Hebbian learning rule. The idea is very intuitive: if a presynaptic neuron fires briefly before the postsynaptic neuron the synaptic weight is increased, conversely, if it fires briefly after the postsynaptic neuron then the synaptic weight is decreased. This extends the classical Hebbian postulate by allowing not only to increase the weights but also decrease them. This learning rule has been derived from empirical observations on biological neurons [82, 83]. A typical expression for this rule is

$$\Delta W_{ij} = \begin{cases} A_+(W_{ij}) \exp\left(-\frac{|t_{pre}-t_{post}|}{\tau}\right), & t_{pre} - t_{post} \leq 0 \\ A_-(W_{ij}) \exp\left(-\frac{|t_{pre}-t_{post}|}{\tau}\right), & t_{pre} - t_{post} > 0 \end{cases} \quad (2.34)$$

where $A_+(W_{ij}) > 0$, $A_-(W_{ij}) < 0$ and t_{pre}, t_{post} are the presynaptic and postsynaptic

spike times respectively.

The response latency of the postsynaptic neuron is decreased as STDP proceeds, resulting in faster neural processing and depending more on the earlier spikes [84]. There are several applications that have used STDP to model the tuning of synaptic connections in sound localisation [85], learning of spatiotemporal spike patterns [39, 86, 87], suppression of predictable signals [88] among many others (see [31, 89] for more examples).

Some more recent work has combined STDP rules with neural architectures from deep learning such as convolutional neural networks to develop unsupervised methods that extract useful features from real world data and solve classification problems [90, 91]. While the feature extraction happens thanks to the STDP rule, the classification usually requires some classical machine learning supervised method such as support vector machines [76] or a variation of the learning rule which introduces some reward signal [92] although it is possible to only rely on STDP and later identify labels with output neurons [93].

2.4.3 Supervised Learning

In supervised learning a model receives a sequence of input samples $\{x_i \in \mathbb{R}^n : i = 1, 2, \dots, N\}$ and corresponding desired outputs $\{y_i \in \mathbb{K} : i = 1, 2, \dots, N\}$ where $\mathbb{K}$ is either $\mathbb{N}$ (classification problem) or $\mathbb{R}$ (regression problem). The goal is to produce the correct output y_j for a never seen before input x_j [72].

We usually distinguish two disjoint sets of data, the *training set* $\mathcal{D}_{train} = \{(x_i, y_i) : i = 1, 2, \dots, N_{train}\}$ that we use to train our model and the *testing set* $\mathcal{D}_{test} = \{(x_i, y_i) : i = 1, 2, \dots, N_{test}\}$ which is never used during training and its sole purpose is to measure the performance of the training procedure. Training means finding the set of parameters that minimise the error between the model predictions and

the desired output. This usually implies minimising a loss function $\mathcal{L}_\theta(\mathcal{D}_{train})$ that depends on the model parameters θ .

Models often have a different set of parameters that often cannot be optimised and must be chosen a priori. These are called hyperparameters. We can use a third disjoint set, the *validation set* $\mathcal{D}_{val} = \{(x_i, y_i) : i = 1, 2, \dots, N_{val}\}$ to tune them.

We briefly review some of the most important supervised algorithms on spiking neural networks which we can divide into two kinds of supervised optimisation algorithms: gradient based and gradient-free algorithms. For a comprehensive survey refer to [94, 89, 95, 11].

Gradient Based Algorithms

Most successful supervised models are neural networks consisting of several layers such as the one defined in 2.4.1 and other more complicated topologies including Convolutional Neural Networks [96], LSTMs [97] and Transformers [98]. Networks are commonly trained using some variation of the stochastic gradient descent algorithm which consists in evaluating the loss function for a partition of the training samples $\mathcal{B}_{train} \subseteq \mathcal{D}_{train}$, backpropagating the gradient of the loss function with respect to the network parameters to all layers and updating the parameters in the gradient direction

$$\theta[n+1] = \theta[n] - \eta \nabla_\theta \mathcal{L}_\theta(\mathcal{B}_{train}) \quad (2.35)$$

This process is repeated iteratively for all partitions. After iterating through the whole training set a new partition is created and the iteration is resumed. This process is repeated for several *epochs* (full training set iterations).

Variations to the stochastic gradient descent algorithm usually involve modifying

the update equation 2.35 to yield faster and better convergence using a history of previous gradient updates. Some of the most popular algorithms include RMSProp [99] and Adam [100].

From a biological point of view gradient based algorithms are problematic. The main issue is the so called ‘weight transport problem’ which arises from the need of a symmetric backward weight for each forward weight in order to backpropagate the gradient. It has been shown that for simple problems, symmetric weights are not necessary but random weights also work, thus partially solving the problem [101, 102].

Even if one does not care about biological plausibility, there is still a technical issue regarding the derivative of spikes. Spiking neuron models often communicate through streams of Dirac delta functions (spike trains) which do not have a well defined derivative. This problem has been circumvented in several ways including: encoding the outputs in spike times (temporal coding networks), using a smooth spiking model, using the membrane potential to backpropagate, optimising the likelihood of spike trains, converting artificial networks into spiking networks, and using a surrogate derivative. More recently it has been shown that it is possible to compute exact gradients on networks of spiking neurons via the adjoint method [12] with some limitations that have been further corrected very recently [103]. We briefly review some of these methods here.

Temporal coding algorithms and Widrow-Hoff learning rule methods. The first supervised algorithm to train SNNs [104] bypassed the need of computing the derivative of a spike by using a Spike Response Model (SRM) which allows to take derivatives of the activations with respect to the spike times. This was later improved for faster convergence in [105]. However, these algorithms had several limitations including constraining the number of output spikes to one per neuron. Subsequent improvements to this method have allowed to add more spikes [106, 107, 108, 109].

Other important work includes the Tempotron [110] and its variants [111, 112, 113] and the Chronotron (E-learning rule) [114].

Another type of algorithms were derived from the Widrow-Hoff rule. The first algorithm that used this rule was the ReSuMe algorithm [115] which is gradient free. Intuitively, this rule updates the weights according to the product of the error signal (desired output minus network output) multiplied times the network activity. Some variants of this algorithm include the Chronotron (I-learning rule) [114], SPAN [116], PSD [117], MPDP [118], FILT [119] and others [120, 121].

Most of these algorithms have only been shown to be successful in solving simple tasks like spike pattern matching, classification of artificially generated spike patterns, the XOR problem and simpler datasets like the Iris [122] or MNIST dataset [123]. Whether this is because of the limitation of the methods or the computational capabilities at the time of these works is not clear.

Probabilistic models. Adding noise to the spike generation allows to compute the likelihood of spike trains smoothly. Consequently, it makes it possible to define gradients on expectation values. This was done first for the single layer case [124] and later in the multilayer case [125]. This approach was also used in the context of variational learning [126, 127, 128].

Membrane backpropagation based methods. Often in detailed spiking models or two-dimensional models (e.g. Hodgkin-Huxley, Morris-Lecar and FitzHugh-Nagumo models) spiking is simply a fast but smooth increase in the membrane potential (soft threshold) thus presenting no challenge to computing the gradient. A model like this was successfully trained in [10]. However, soft threshold models do not have the binary spike propagation feature of SNNs which is particularly interesting for its application on low energy neuromorphic hardware [95]. An alternative to soft spiking models consists on using the membrane potential as the differentiable activation function instead of the spikes while still propagating spikes [129, 130].

ANN to SNN conversion. To get around the difficulties of gradient descent in spiking networks, it is possible to convert a pretrained artificial neural network (ANN) into an SNN by adjusting the weights and parameters of the spiking neurons with the aim of replicating the input-output mapping of the original ANN in the SNN. This is usually achieved by either taking an already trained ANN and converting it to an SNN [131, 132, 133, 134, 135] or by training a constrained ANN to better fit the conversion to SNN [136]. Network conversion has achieved the highest levels of accuracy for SNNs when compared to other benchmarks. Nevertheless, this process has several caveats. Firstly, not all activation functions and neural architectures can be easily converted (e.g. max pooling layers are substituted by average pooling layers). Secondly, since ANN activations take continuous values, it is usually necessary to resort to rate-based coding on the SNN. This results in less energy efficient networks than if the the original ANN was used in the first place[137].

Surrogate gradients. Surrogate gradient methods introduce a smooth function that approximates the spiking function (the Heaviside step function) but only in the computation of the gradients. Thus, the network still propagates spikes in the forward pass but it backpropagates continuous values. Figure 2.6 shows several of these approximations. This allows to treat SNNs as a special case of recurrent neural networks (RNNs) and apply conventional algorithms such as Backpropagation Through Time (BPTT) or forward methods like Real-Time Recurrent Learning (RTRL) [11]. One of the first uses of surrogate gradient approximates the gradient using a piecewise linear function [138]. This was also used later to implement trained SNNs on neuromorphic hardware [139]. Other surrogate functions have been successfully used [102, 140, 141, 142, 143, 144] and more recently it has been shown that SNNs can robustly learn under a wide range of surrogate functions [145]. This method is employed in subsequent chapters which provide more details on its implementation (see section 3.3 and Appendix B.1).

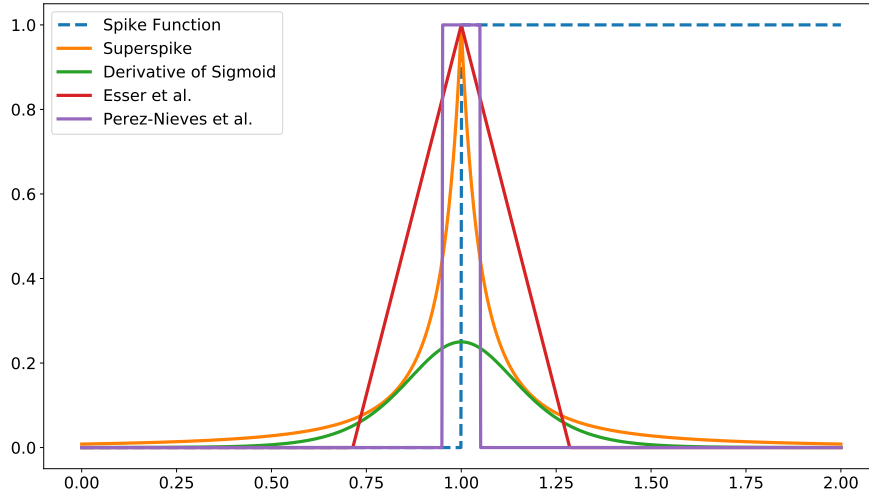


Figure 2.6: Spike function $H(v - V_{th})$ with threshold $V_{th} = 1$ (dashed) and different types of surrogate derivatives used in the literature including: Superspike $\frac{1}{1+\beta|v-V_{th}|}$ [102], derivative of sigmoid $\sigma(v - V_{th})(1 - \sigma(v - V_{th}))$ with $\sigma(x)$ being the sigmoid function [145], piecewise linear $\max(0, 1 - \beta|v - V_{th}|)$ [146] and piecewise constant $H(v - (V_{th} - B_{th})) - H(v - (V_{th} + B_{th}))$ [147]

Gradient-free Algorithms

Gradient-free supervised algorithms to train SNNs are more diverse than gradient-based algorithms and use quite different approaches to solving the problem of training spiking neural networks. Here we briefly review some of the most prominent ones.

Liquid-state machines. Liquid-state machines were first introduced in [148] following the hypothesis that through evolution the mammalian cortex found a structural motif that can be replicated and modified to serve different purposes. This structure has often been called minicolumn or cortical microcircuit and consists of a few hundred sparsely and recurrently connected neurons. Input signals are injected into this network (also called a reservoir). A set of linear units is connected to the reservoir and it is trained using linear regression to match a desired output. The reservoir usually does not learn (although sometimes it is also implemented with

STDP rules) but serves as a non-linear transformation of the input data [149, 89].

FORCE training. The FORCE method originates from the field of reservoir computing. It leverages the complex dynamics of a high dimensional dynamical system (e.g. a recurrent SNN) to perform computations. Originally applied to rate neuron models exclusively [150] in more recent years it has also been successfully applied to spiking networks [151, 152]. In FORCE training the aim is to approximate a teaching signal with a linearly decoded signal from the network such as the synaptic currents. The weights are usually updated using the recursive least squares (RLS) algorithm [153]. Details for this method are provided in Chapter 3.

2.4.4 Reinforcement Learning

The reinforcement learning (RL) paradigm is rather different to the previous two. In this case, there is an *agent* and an *environment*. The aim of the agent is to act on the environment to maximise a cumulative reward. The interaction usually consists in the following iteration: the agent sees a (possibly partial) observation o_t of the state of the environment s_t and takes an action a_t which affects the environment prompting it to update its state to s_{t+1} with its corresponding updated observation o_{t+1} . For each environment transition the agent also receives a reward r_t .

States fully describe the environment at a given time. These are typically represented by a vector or a higher order tensor of real values. When an environment is fully observed, agents have access to the full state at each time step otherwise the environment is partially observed. Environments can be either deterministic $s_{t+1} = f(s_t, a_t)$ or stochastic $s_{t+1} \sim P(\cdot | s_t, a_t)$.

Action spaces may be discrete (only a finite number of actions are available, i.e. an action is a vector of discrete values), continuous (actions are real-valued vectors) or mixed (combination of the two). Agents choose actions according to a *policy* which

may be either deterministic $a_t = \mu(o_t)$ or stochastic $a_t \sim \pi(\cdot|o_t)$.

Rewards are scalar values that are most often deterministically generated as a result of the transition (s_t, a_t, s_{t+1}) , thus $r_t = r(s_t, a_t, s_{t+1})$. Sometimes rewards may only depend on the current state or the current state and current action. Agents often aim to maximise a *return* function which is often the finite-horizon undiscounted return $R = \sum_{t=0}^T r_t$ or the infinite-horizon discounted return $R = \sum_{t=0}^{\infty} \gamma^t r_t$ with $\gamma \in (0, 1)$. In order to do this the agent must acquire data by interacting with the environment to generate trajectories $\tau = (s_0, a_0, s_1, a_1, \dots)$ and rewards $(r_0, r_1, \dots)$ which can then be used to learn an optimal policy. Thus, the aim of an agent is to find an optimal policy so that the total expected return $\mathbb{E}_{\tau \sim \pi}[R(\tau)]$ is maximised.

Reinforcement learning is a natural learning paradigm for biologically inspired models because many animal behaviours are learned by interacting with the environment in this way. For example, we can train mice to follow complex sequences of actions by rewarding them when performing the desired behaviour. In the computational neuroscience literature this has led to theoretical models that attempt to explain how can such learning mechanisms be implemented in the neural circuits while fitting as much as possible experimental data. The most notable result of this are three-factor learning rules. More recently, and inspired by the huge success of deep reinforcement learning [154, 6, 155, 9] there is also an effort to use similar techniques on spiking neural networks with the aim of faster and more efficient learning rather than biological plausibility.

While we do not use reinforcement learning techniques in this thesis, we still believe it is important to give a very brief review of the literature of RL in SNNs due to its biological importance and recent success.

Three-factor learning rules. The aim of most three-factor learning rules studies is to find synaptic plasticity rules that could potentially model learning in biological neural circuits. Three-factor learning rules [156, 157] combine unsupervised Hebbian

learning (which consists of two factors, pre and post synaptic activity) with a third factor which encodes the reward signal. This allows to potentiate synapses that do lead to the desired outcome and depress those that do not. Often this third factor attempts to model the influence of a neuromodulator such as dopamine. The general rule has the form

$$\frac{dW_{ij}}{dt} = f(W_{ij}, \hat{v}_i, \hat{v}_j, M) \quad (2.36)$$

where M is the extrinsic signal that encodes the reward and is not generated by either the pre or post synaptic neurons. Some examples of this rules include R-max rules which are ultimately derived from policy gradient methods [158, 159, 160], temporal-difference based rules [161, 162, 163] and R-STDP rules which are based on heuristic considerations [159, 164, 160, 165] (see [156, 157] for an in-depth review).

Deep reinforcement learning on SNNs. Deep reinforcement learning on SNNs aims to adapt RL techniques that have proven to be successful in ANNs to SNNs. This allows to solve larger more complex problems that previous three-factor rules could not solve. Some of these include solving control problems in MuJoCo environments [166], grid-world games [167] and Atari games [168]. These works tend to aim to use SNNs due to its lower computational an energy cost leading to consuming up to 140x less energy [166]. Most of these results are based on jointly applying RL algorithms such as DDPG [169] or PPO [170] with deep SNN training techniques such as those reviewed in the previous section 2.4.3.

Chapter 3

Neural Heterogeneity Promotes Robust Learning

This chapter explores the impact of using heterogeneous parameters on spiking neural networks trained using surrogate gradient descent and the FORCE method. It challenges common network architectures where all neurons are identical units by introducing different kinds of neural variability and systematically studying its effects. The results support the idea that heterogeneity is a desirable and important property in biological systems with an actual role in information processing in nervous systems.

3.1 Introduction

The study of complex systems often starts with a simplified model of the object of study to make easier its understanding. In physics, the study of gases is often simplified by considering molecules as point masses that interact with perfect elastic collisions. In neuroscience, biological neurons are often modelled as point neurons which are governed by a few differential equations and a threshold mechanism.

The purpose of a model is to capture the essential features of a real system, while omitting details that are not important or that are too difficult to model accurately. Understanding the limits of a model is crucial to avoid inaccurate predictions. For instance, the ideal gas model fails when considering gases at very high temperatures. Similarly, it is desirable to know to what extent point neuron networks are good enough to model the brain. However, to this date, we hardly know what the cost of these simplified models actually is and it is largely unexplored whether adding more complexity to point neuron network models could have computational benefits. In this chapter we aim to partly answer this question by challenging the common practice in computational neuroscience of considering all neurons as highly simplified identical units.

The brain is known to be deeply heterogeneous at all scales [171], however it is still not known whether this heterogeneity plays an important functional role or if it is just a byproduct of noisy developmental processes and contingent evolutionary history. A number of hypothetical roles have been suggested (reviewed in [172]), including in efficient coding [173, 174, 175, 176, 177, 178, 179], reliability [180], working memory [181], and functional specialisation [182]. However, previous studies have largely used simplified tasks or networks, and it remains unknown whether or not heterogeneity can help animals solve complex information processing tasks in natural environments. Recent work has allowed us, for the first time, to train realistic spiking neural networks to carry out these tasks at a high level of performance, using methods derived from machine learning. However, learning in spiking neurons has often been limited to learning synaptic weights exclusively leaving other neural and synaptic parameters to be homogeneous and static [11, 152, 140]. The spiking dynamics of biological neural networks are not only governed by the distribution of synaptic weights, but also by all other neural and synaptic parameters. Thus, it is only natural to study the functional role of these parameters within spiking neural networks.

Some recent work suggests that there may be an advantage on training other neural and synaptic parameters in addition to weights. For instance, in [140] both synaptic weights and delays are trained, in [183] it is found that neurons with adaptive threshold are crucial in order to achieve the best performance and, more recently, in [184] synaptic filters dynamics are trained in addition to synaptic weights. Other works have suggested that the heterogeneity present in the brain may play several important functional roles [148, 185, 181, 180, 186, 179], notably heterogeneous tuning may be important for robust population coding of complex stimuli [176, 187]. There is also work that suggests that heterogeneity plays a crucial role in neural information processing [180], enhancing the information carrying capacity across nervous systems [172] and results on a higher computational power [182]. Despite these advances, the advantages of heterogeneity of parameters in SNNs have yet not been systematically investigated.

We used two different learning models [152, 11] to investigate the effect of introducing heterogeneity in the time scales of neurons when performing tasks with realistic and complex temporal structure. We found that heterogeneity of synaptic and membrane time constants improves training performance for tasks where information is mainly encoded in the spike times while seems to have almost no impact in tasks where information is encoded in the spatial distribution of spikes. We found that heterogeneity emerges even when time constants are initialised homogeneously beyond simply spreading around its initial value and that the distributions of parameters found qualitatively match the experimental data. We also found that these improvements were not due to a better fine tuning of network parameters but were in fact due to the heterogeneity of parameters itself. Finally, we also found that heterogeneity can make up for mistuning in the initialisation of time constants, thus removing the need of performing an otherwise costly hyperparameter search.

In brief heterogeneity improves the overall performance, makes learning more stable

and robust and networks learn neural parameter distributions that match experimental observations, suggesting that the heterogeneity observed in the brain may be a vital component of its ability to adapt to new environments.

3.2 Results

3.2.1 Time scale heterogeneity improves learning on tasks with rich temporal structure

We investigated the role of neural heterogeneity in task performance by training recurrent spiking neural networks to classify visual and auditory stimuli with varying degrees of temporal structure. The model used three layers of spiking neurons: an input layer, a recurrently connected layer, and a readout layer used to generate predictions (Fig. 3.1A), a widely used minimal architecture (e.g. Maass et al. [148], Neftci et al. [11]). Heterogeneity was introduced by giving each neuron an individual membrane and synaptic time constant. We compared four different conditions: initial values could be either homogeneous or heterogeneous, and training could be either standard or heterogeneous (Fig. 3.1B). In more detail, time constants were either initialised with a single value (homogeneous initialisation), or randomly according to a gamma distribution (heterogeneous initialisation). In both types of training, the parameters of the models were optimised using surrogate gradient descent [11]. Synaptic weights were trainable in both standard and heterogeneous training regimes, while time constants were either held fixed at their initial values in the standard training regime, or could be modified in the heterogeneous training regime.

We used five different datasets with varying degrees of temporal structure. Neuromorphic MNIST (N-MNIST) [188], xiao2017fashion (F-MNIST) [189], and the

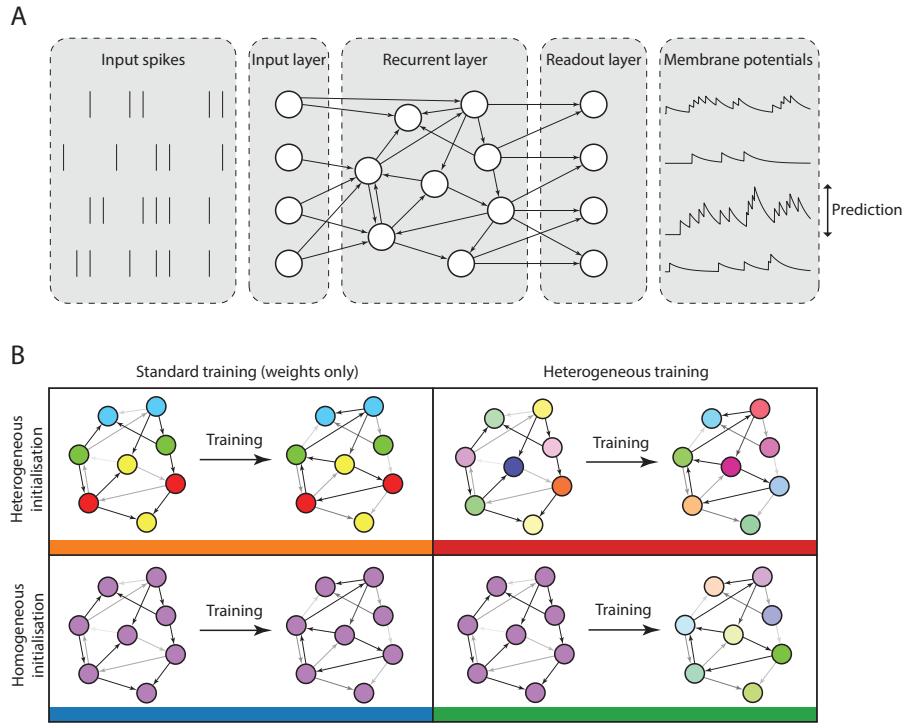


Figure 3.1: **Diagram of network architecture and training configurations.**

A. Model architecture. A layer of input neurons emits spike trains into a recurrently connected layer of spiking neurons which is followed by a readout layer. **B.** Configurations. Training can be either standard (only the synaptic weights are learned) or heterogeneous (the synaptic weights and membrane and synaptic time constants are learned). The initialisation can be homogeneous (all synaptic and membrane time constants are initialised to the same value) or heterogeneous (synaptic and membrane time constants are randomly initialised for each neuron by sampling them from a given probability distribution).

DVS128 Gesture dataset [190] feature visual stimuli, while the Spiking Heidelberg Digits (SHD) and Spiking Speech Commands (SSC) datasets [191] are auditory. N-MNIST and DVS128 use a neuromorphic vision sensor to generate spiking activity, by moving the sensor with a static visual image of handwritten digits (N-MNIST) or by recording humans making hand gestures (DVS128). F-MNIST is a dataset of static images that is widely used in machine learning, which we converted into spike times by treating the image intensities as input currents to model neurons, so that higher intensity pixels would lead to earlier spikes, and lower intensity to later spikes. Both SHD and SSC use the same detailed model of the activity of bushy cells in the cochlear nucleus developed in Cramer et al. [191], in response to

spoken digits (SHD) or commands (SSC). Their model consists of standard components from the auditory modelling literature, including a hydrodynamic basilar membrane, transmitter pool and inhomogeneous Poisson process hair cell, and leaky integrate-and-fire bushy cell. Of these datasets, N-MNIST and F-MNIST have minimal temporal structure, as they are generated from static images. DVS128 has some temporal structure as it is recorded motion, but it is possible to perform well at this task by discarding the temporal information. The auditory tasks SHD and SSC by contrast have very rich temporal structure. In all cases we used the train/test split suggested by the corresponding dataset authors to compare our performance with previous baselines fairly.

We found that heterogeneity in time constants had a profound impact on performance on those training datasets where information was encoded in the precise timing of input spikes (Table 3.1, Fig. 3.2A). On the most temporally complex auditory tasks, accuracy improved by a factor of around 15-20%, while for the least temporally complex task N-MNIST we saw no improvement at all. For the gesture dataset DVS128 we can identify the source of the (intermediate) improvement as the heterogeneous models being better able to distinguish between spatially similar but temporally different gestures, such as clockwise and anticlockwise versions of the same gesture (see Fig. 3.3). This suggests that we might see greater improvements for a richer version of this dataset in which temporal structure was more important.

We verified that our results were due to heterogeneity and not simply to a better tuning of time constants in two ways. Firstly, we performed a grid search across all homogeneous time constants for the SHD dataset and used the best values for our comparison. Secondly, we observe that the distribution of time constants after training is very similar and heterogeneous regardless of whether it was initialised with a homogeneous or heterogeneous distribution (Fig. 3.2B), indicating that the heterogeneous distribution is optimal.

Initialisation-Training	N-MNIST	F-MNIST	DVS128	SHD	SSC
Homog-Standard	97.4 $\pm$ 0.0	80.1 $\pm$ 7.4	76.9 $\pm$ 0.8	71.7 $\pm$ 1.0	49.7 $\pm$ 0.4
Heterog-Standard	97.5 $\pm$ 0.0	87.9 $\pm$ 0.1	79.5 $\pm$ 1.0	73.7 $\pm$ 1.1	53.7 $\pm$ 0.7
Homog-Heterog	96.6 $\pm$ 0.2	79.7 $\pm$ 7.4	81.2 $\pm$ 0.8	82.7 $\pm$ 0.8	56.6 $\pm$ 0.7
Heterog-Heterog	97.3 $\pm$ 0.1	87.5 $\pm$ 0.1	82.1 $\pm$ 0.8	81.7 $\pm$ 0.8	60.1 $\pm$ 0.7
Chance level	10.0	10.0	10.0	5.0	2.9

Table 3.1: **Testing accuracy percentage over different datasets and training methods** Effect of initialisation and training configuration on performance, on datasets of increasing temporal complexity. Initialisation can be homogeneous (all time constants the same) or heterogeneous (random initialisation), and training can be standard (only synaptic weights learned) or heterogeneous (time constants can also be learned). N-MNIST and F-MNIST are static image datasets with little temporal structure, DVS128 is video gestures, and SHD and SSC are temporally complex auditory datasets.

Introducing heterogeneity allows for a large increase in performance at the cost of only a very small increase in the number of parameters (0.23% for SHD, because we have added some neuron-specific parameters, but not added any synapse-specific ones, and the vast majority of parameters are synaptic weights), and without using any additional neurons or synapses. Heterogeneity is therefore a metabolically efficient strategy. It is also a computationally efficient strategy of interest to neuro-morphic computing, because adding heterogeneous time constants to the model adds $O(n)$ to memory use and computation time when simulating the model, while adding more neurons adds $O(n^2)$ (because the model is fully connected, so the number of synaptic weights is proportional to the square of the number of neurons).

Note that it is possible to obtain better performance using a larger number of neurons. For example, Cramer et al. [191] obtained a performance of 83.2% on the SHD dataset without heterogeneity using 1024 neurons and data augmentation techniques, whereas we obtained 82.7% using 128 neurons and no data augmentation. We focus on smaller networks here for two reasons. Firstly, we wanted to systematically investigate the effect of different training regimes, and current limitations of surrogate gradient descent mean that each training session takes several days.

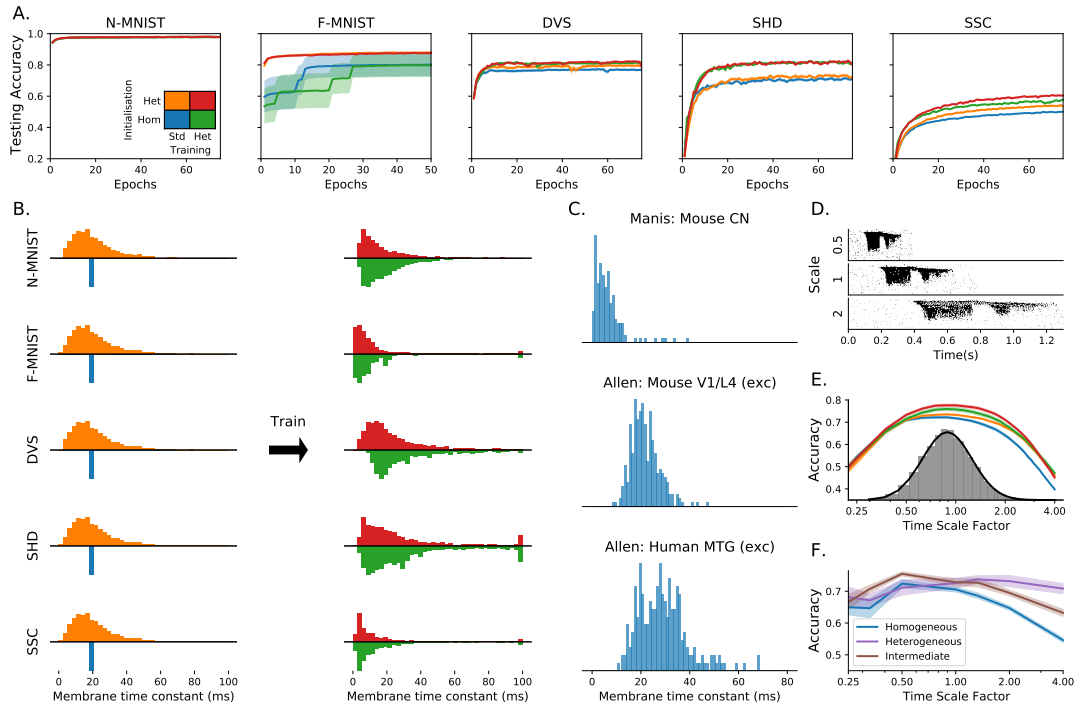


Figure 3.2: Impact of training configuration and temporal structure of the dataset on the testing accuracy, membrane time constant distributions and performance when training at different time scales. **A.** Improvements in accuracy in testing data, for datasets with low (N-MNIST, F-MNIST), intermediate (DVS) and high (SHD) temporal complexity. Shaded areas correspond to standard error of the mean over 10 trials (which in some cases is too small to be visible). Initialisation can be homogeneous (blue/green) or heterogeneous (orange/red), and training can be standard, weights only (blue/orange) or heterogeneous including time constants (green/red). Heterogeneous configurations achieve a better test accuracy on the more temporally complex datasets. Heterogeneous initialisation also results in a more stable and robust training trajectory for F-MNIST, leading to a better performance overall. **B.** Membrane time constant distributions before (left) and after (right) training for each dataset. Histograms above the axis represent heterogeneous initialisation, and below the axis homogeneous initialisation. In the case of standard training (weights only), the initial distribution (left) is the same as the final distribution of time constants after training. **C.** Experimentally observed distributions of time constants for (top to bottom): mouse cochlear nucleus, multiple cell types (172 cells)[192, 193]; mouse V1 layer 4, spiny (putatively excitatory) cells (164 cells)[194, 195]; human middle temporal gyrus, spiny cells (236 cells)[194, 195]. **D.** Raster plot of input spikes from a single sample of the SHD dataset (spoken digits) at three different time scales. **E.** Accuracy on the SHD dataset after training on a variety of time scales (randomly selected from the grey distribution) for the four configurations described in (A). **F.** Accuracy on the SHD dataset when the initial distribution of time constants is tuned for time scale 1.0, but the training and testing is done at different time scales.

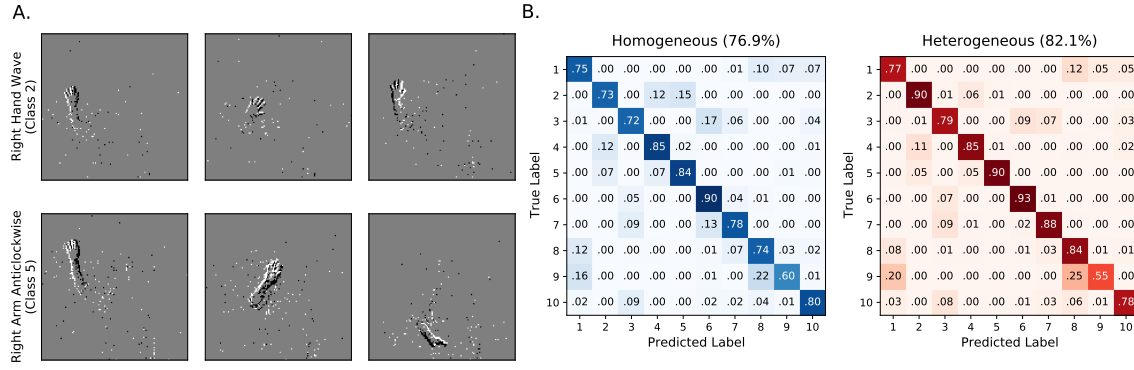


Figure 3.3: **A.** Visualisation of two samples of the DVS128 gesture dataset. Each frame spans a 5ms window with 200ms between frames. To distinguish these gestures we need to integrate over tens of milliseconds. **B.** Confusion matrices of the DVS128 gesture dataset under the fully homogeneous (left) and fully heterogeneous (right) configurations. Class 2 (*Right hand wave*) is often incorrectly classified as 4 (*Right hand clockwise*) and 5 (*Right hand counter clockwise*) under the homogeneous configuration but not under the heterogeneous one.

Secondly, with larger numbers of neurons, performance even without heterogeneity can in some cases approach the ceiling on these tasks (which are still simple in comparison to those faced by animals in real environments), making it more difficult to see the effect of different architectures. In the case of SSC dataset however, even with this limitation to small networks, heterogeneity confers such an advantage that our results are state of the art (for spiking neural networks) by a large margin.

We also tested the effect of introducing heterogeneity of other neuron parameters, such as the firing threshold and reset potential, but found that it made no appreciable difference. This was because for our model, changing these is almost equivalent to a simple scaling of the membrane potential variable. By contrast, Bellec et al. [183] found that introducing an adaptive threshold did improve performance, presumably because it allows for much richer temporal dynamics (in line with earlier findings such as [196]).

3.2.2 Predicted time constant distributions match experimental data

In all our tasks, the distributions of time constants after training approximately but not exactly fit a log normal or gamma distribution (with different parameters for each task). They are also consistent across different training runs (see Fig. 3.4 and Fig. 3.5) and initial distributions (see Fig. 3.6 and Fig. 3.7), suggesting that the learned distributions may be optimal. Using publicly available datasets including time constants recorded in large numbers of neurons in different animals and brain regions [192, 193, 194, 195], we found similar distributions to those we predicted (Fig. 3.2C). This is striking, as our model is optimised for a single, relatively simple task, while animals in real environments face a range of tasks, each of which is more difficult than our model tasks. The parameters for the experimentally observed distributions are different for each animal and region, just as for different tasks in our simulations. Interestingly, the distribution parameters are also different for each cell type in the experimental data, a feature not replicated in our simulations as all cells are identical. This suggests that introducing further diversity in terms of different cell types may lead to even better performance.

3.2.3 Generalisation across time scales

Sensory signals such as speech and motion can be recognised across a range of speeds by humans and animals. We tested the role of heterogeneity in learning a circuit that can function across a wide range of speeds. We augmented the SHD spoken digits datasets to include faster or slower versions of the samples, multiplying all spike times by a temporal scale as an extremely simplified model that captures a part of the difficulty of this task (Fig. 3.2D). During training, temporal scales were randomly selected from a distribution roughly matching human syllabic rate distributions

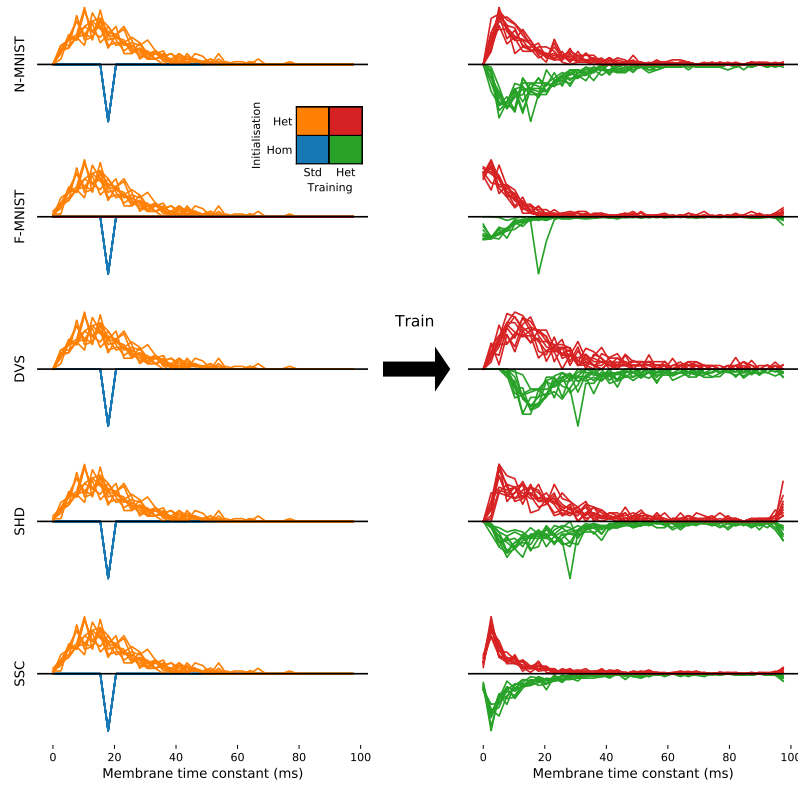


Figure 3.4: Breakdown of membrane time constant distribution for each trial for consistency checking on each dataset.

[197]. The heterogeneous configurations performed as well or better at all time scales (Fig. 3.2E), and in particular were able to generalise better to time scales outside the training distribution (e.g. an accuracy of 47% close to a time scale of 4, around 7% higher than the homogeneous network, where chance performance would be 5%). Heterogeneous initialisation alone was sufficient to achieve this better generalisation performance, while time constant training improved the peak performance but gave no additional ability to generalise.

3.2.4 Robustness against mistuned learning

We tested the hypothesis that heterogeneity can provide robustness with two experiments where the hyperparameters were mistuned, that is where the initial distributions and learning parameters were chosen to give the best performance for one

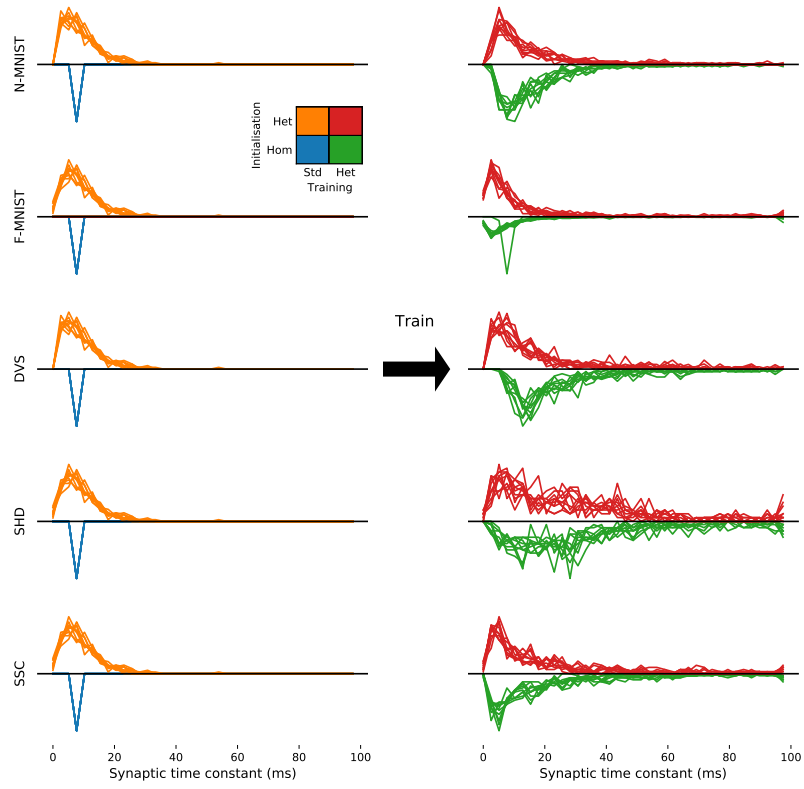


Figure 3.5: Breakdown of synaptic time constant distribution for each trial for consistency checking on each dataset.

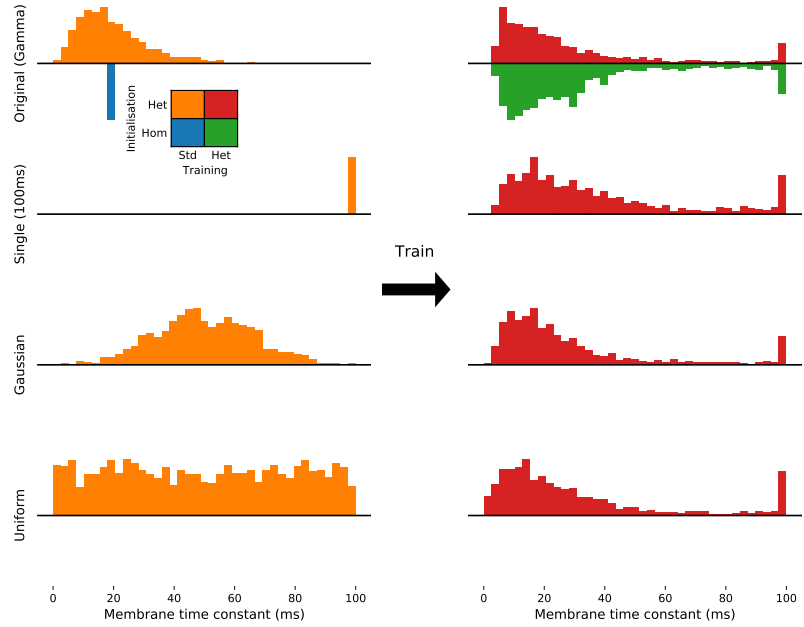


Figure 3.6: Membrane time constant distributions before (left) and after (right) training for SHD dataset. Each row represents a different initialisation of the time constants.

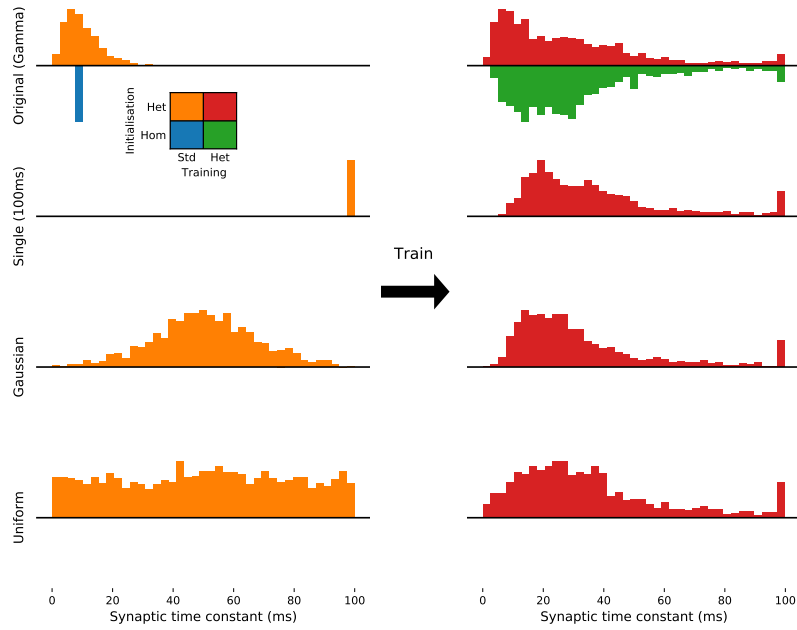


Figure 3.7: Membrane time constant distributions before (left) and after (right) training for SHD dataset. Each row represents a different initialisation of the time constants.

distribution, but the actual training and testing is done on a different distribution.

In the first experiment, we took the augmented SHD spoken digits dataset from the previous section, selected the hyperparameters to give the best performance at a time scale of 1, but then trained and tested the network at a different time scale (Fig. 3.2F). We used training of weights only, since allowing retraining of time constants lets the network cancel out the effect of changing the time scale. With a homogeneous or narrow heterogeneous initial distribution (blue line in Fig. 3.2F), performance falls off for time scales far from the optimal one, particularly for larger time scales (slower stimuli). However, a wide heterogeneous initial distribution (purple line in Fig. 3.2F) allows for good performance across all time scales tested, at the cost of a slightly lower peak performance at the best time scale. We tested whether or not this was solely due to the presence of a few neurons with long time constants by introducing an intermediate distribution where the majority of time constants were the same as the homogeneous case, with a small minority of much longer time constants (brown line in Fig. 3.2F). The performance of this distribution was inter-

mediate between the homogeneous and heterogeneous distributions, a pattern that is repeated in our second experiment below.

In the second experiment, we switched to a very different learning paradigm, FORCE training of spiking neural networks [152] to replay a learned signal, in this case a recording of a zebra finch call (Fig. 3.8A; from Blättler and Hahnloser [198]). This method does not allow for heterogeneous training (tuning time constants), so we only tested the role of untrained heterogeneous neuron parameters. We tested three configurations: fully homogeneous (single 20ms time constant as in the original paper); intermediate (each neuron randomly assigned a fixed fast 20ms or slow 100ms time constant); or fully heterogeneous (each neuron randomly assigned a time constant drawn from a gamma distribution).

Nicola and Clopath [152] showed that network performance is highly dependent on two hyperparameters (G and Q in their paper). We therefore tuned these hyperparameters for a network of a fixed size ($N = 1000$ neurons) and ran the training and testing for networks of different sizes (Fig. 3.8B). As the network size started to diverge, the homogeneous network began to make large errors, while the fully heterogeneous network was able to give low errors for all network sizes. The intermediate network was able to function well across a wider range of network sizes than the fully homogeneous network, but still eventually failed for the largest network sizes. At these large network sizes, the homogeneous network neurons fire at or close to their maximal rate and training cannot take place, leading to poor performance (Fig. 3.8C). The robustness of the heterogeneous version of the network can be measured by the area of the hyperparameter space that leads to good performance (Fig. 3.8D). Adding partial or full heterogeneity leads to an improvement in learning for all points in the hyperparameter space, again suggesting that it can improve robustness of learning in a wide range of situations.

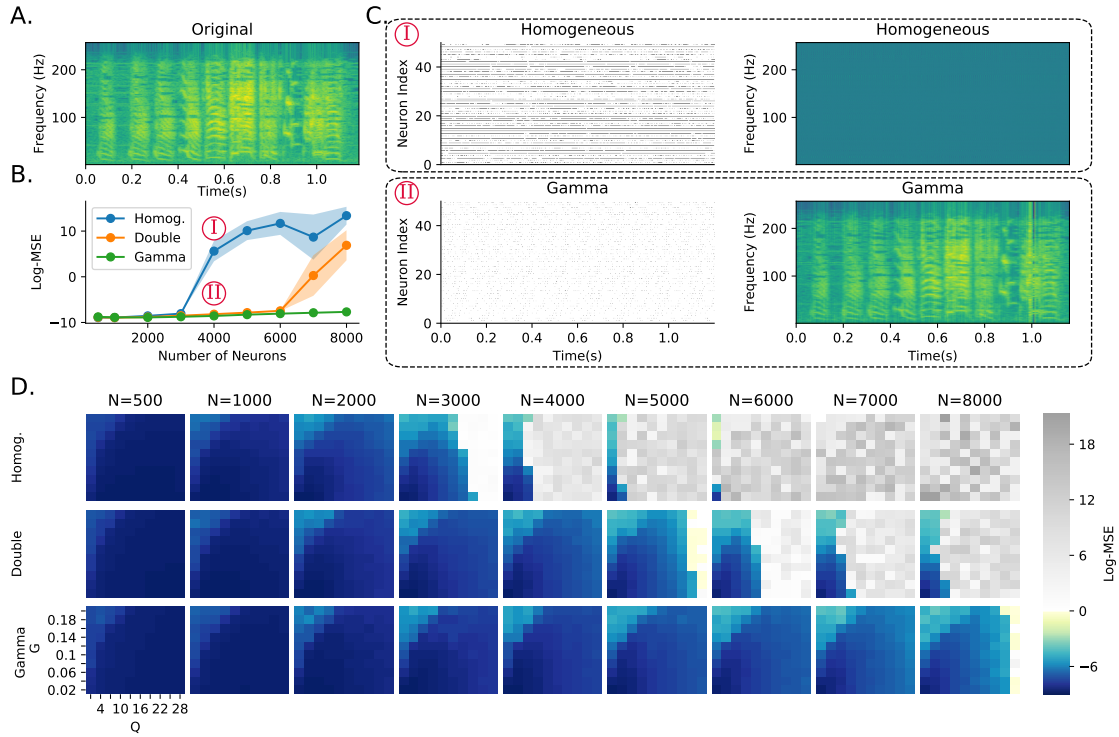


Figure 3.8: **Robustness to learning hyperparameter mistuning.** **A.** Spectrogram of a zebra finch call. The network has to learn to reproduce this spectrogram, chosen for its spectrotemporal complexity. **B** Error for three networks at different network sizes (hyperparameters G and Q were chosen to optimise performance at $N = 1000$ neurons as given in Table 3.4). Networks are fully homogeneous (*Homog.*); intermediate, where each neuron is randomly assigned slow or fast dynamics (*Double*); or fully heterogeneous, where each neuron has a random time constant drawn from a gamma distribution (*Gamma*). **C.** Raster plots of 50 neurons randomly chosen, and reconstructed spectrograms under fully homogeneous and fully heterogeneous (*Gamma*) conditions for $N = 4000$ neurons as indicated in (B). **D.** Reconstruction error. Each row is one of the three configurations shown as lines in (B). Each column is a network size. The axes of each image give the learning hyperparameters (G and Q). Grey pixels correspond to log mean square error above 0, corresponding to a complete failure to reconstruct the spectrogram. The larger the coloured region, the more robust the network is, and the less tuning is required.

3.3 Methods

3.3.1 Neuron and synaptic models

We use the Leaky Integrate and Fire (LIF) neuron model in all our simulations. In this model the membrane potential of the j -th neuron in the l -th layer $v_j^{(l)}(t)$ varies over time following Equation (3.1).

$$\tau_m \frac{dv_j^{(l)}(t)}{dt} = -(v_j^{(l)}(t) - V_0) + RI_j^{(l)}(t) \quad (3.1)$$

Here, τ_m is the membrane time constant, V_0 is the resting potential and $I_j^{(l)}(t)$ is the input current. When the membrane potential reaches the threshold value V_{th} a spike is emitted, $v_j^{(l)}(t)$ resets to the reset potential V_r and then enters a refractory period that lasts t_{ref} seconds where the neuron cannot spike. For the remaining we assume $R = 1$.

Spikes emitted by the i -th neuron in layer $l-1$ at a finite set of times $\{t_i^{(k)}\}$ can be formalised as a spike train $s_i^{(l-1)}(t)$ defined as in Equation (3.2)

$$s_i^{(l-1)}(t) = \sum_k \delta(t - t_i^{(k)}) \quad (3.2)$$

The input current $I_j^{(l)}(t)$ is obtained from the spike trains of all presynaptic neurons connected to neuron j following Equation (3.3)

$$\tau_s \frac{dI_j^{(l)}(t)}{dt} = -I_j^{(l)}(t) + \sum_i W_{ij}^{(l)} s_i^{(l-1)}(t) + \sum_k V_{kj}^{(l)} s_k^{(l)}(t) \quad (3.3)$$

Here τ_s is the synaptic time constant, $W_{ij}^{(l)}$ is the feed-forward synaptic weight from neuron i in layer $l-1$ to neuron j in layer l and $V_{ij}^{(l)}$ is the recurrent weight from

neuron i in layer l to neuron j in layer l .

Thus, a LIF neuron is fully defined by six parameters $\tau_m, \tau_s, V_{th}, V_0, V_r, t_{ref}$ plus its synaptic weights $W_{ij}^{(l)}$ and $V_{ij}^{(l)}$. We refer to these as the neuron parameters and weights respectively.

Since we are considering the cases where these parameters may be different for each neuron in the population we should actually refer to $\tau_{m,j}, \tau_{s,j}, V_{th,j}, V_{0,j}, V_{r,j}, t_{ref,j}$. However, for notational simplicity we will drop the j subscript and it will be assumed that these parameters can be different for each neuron in a population.

3.3.2 Neural and synaptic model discretisation

In order to implement the LIF model in a computer it is necessary to discretise it. We follow the discretisation in Cramer et al. [191] and 2.3.2. Here we use square brackets to index variables changing in a discrete time setting, so that t refers to a continuous value when written $f(t)$ or an integer when written $f[t]$. We discretise time into multiples of a small time step Δt , so that spikes can only happen at multiples of Δt . With this discretisation, we can approximately solve Equation (3.3) to give

$$I_j^{(l)}(t + \Delta t) = I_j^{(l)}[t + 1] = \alpha I_j^{(l)}[t] + \sum_i W_{ij} s_i^{(l-1)}[t] + \sum_i V_{kj} s_k^{(l)}[t] \quad (3.4)$$

with $\alpha = \exp(-\Delta t / \tau_s)$. Similarly, Equation (3.1) becomes

$$v_j^{(l)}(t + \Delta t) = v_j^{(l)}[t + 1] = \beta v_j^{(l)}[t] + (1 - \beta) \left(V_0 + I_j^{(l)}[t] \right) - (V_{th} - V_r) s_j^{(l)}[t] \quad (3.5)$$

with $\beta = \exp(-\Delta t / \tau_m)$. The last term is added to reset the neuron after spiking and it is often called a ‘soft threshold’ (as opposed to a ‘hard threshold’ which would consist on setting the membrane to V_r).

Finally, the spiking mechanism

$$s_j^{(l)}[t] = \begin{cases} 1, & \text{if } v_j^{(l)}[t] \geq V_{th} \\ 0, & \text{if } v_j^{(l)}[t] < V_{th} \end{cases} \quad (3.6)$$

Notice how the last term in Equation (3.5) introduces the membrane potential resetting. This would only work if we assume that the neuron potential at spiking time was exactly equal to V_{th} . This may not necessarily be the case since a membrane potential update that crossed the threshold may result in $v_i^{(l)} > V_{th}$ and then the resetting mechanism will not set the membrane potential to V_r . However, we found that this has a negligible effect in our simulations.

Surrogate Gradient Descent Training

With the discretisation introduced in the previous section, a spiking layer consists of three cascaded sub-layers: current from Equation (3.4), membrane from Equation (3.5) and spike from Equation (3.6). The current and membrane sub-layers have access to their previous state and thus, they can be seen as a particular case of recurrent neural network (RNN). Note that while each neuron is a recurrent unit since it has access to its own previous state, different neurons in the same spiking layer will only be connected if any of the non-diagonal elements of $\mathbf{V}^{(l)}$ is non-zero. In other words, all spiking neural networks (SNNs) built using this model are RNNs but not all SNNs are recurrent SNNs (RSNNs).

We can cascade L spiking layers to conform a Deep Spiking Neural Network analogous to a conventional Deep Neural Network and train it using gradient descent. However, since Equation (3.6) is non-differentiable, we need to modify the backwards pass as in [11] so that the backpropagation through time (BPTT) algorithm can be used to update the network parameters.

$$\sigma(V_j^{(l)}) = \frac{V_j^{(l)}}{1 + \rho|(V_j^{(l)})|} \quad (3.7)$$

This means that while in the forward pass the network follows a step function as in Equation (3.6), in the backwards pass it follows a sigmoidal function described in Equation (3.7), with a steepness set by $\rho > 0$.

We can now use gradient descent to optimise the synaptic weights $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l-1)} \times n^{(l)}}$ and $\mathbf{V}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l)}}$ as in conventional deep learning. We can also optimise the spiking neuron specific parameters V_{th}, V_0, V_r since they can be seen as bias terms. The time constants can also be indirectly optimised by training α and β which can be seen as forgetting factors.

We apply a clipping function to α and β after every update.

$$\text{clip}(x) = \begin{cases} e^{-1/3}, & \text{if } x < e^{-1/3} \\ 0.995, & \text{if } x > 0.995 \end{cases} \quad (3.8)$$

In order to ensure stability, the forgetting factors have to be less than 1. Otherwise, the current and membrane potential would grow exponentially. Secondly to make the system causal these factors cannot be less than zero. This however, would allow for arbitrarily small time constants and for a finite time resolution Δt leads to constant firing and numerical instability. Thus, we constrain the time constants to be at least $3\Delta t$. This leads to the minimum value being $\exp(-1/3)$. As for the upper bound, we chose a maximum time constant of 100ms (corresponding to $\exp(-\Delta t/100) = 0.995$) informed by the time constants reported in the NeuroElectro database [199] which shows that about 99.5% of the membrane time constants recorded from biological neurons have time constants below this value. The lower bound is also consistent with this database given our simulation time step of 0.5ms. Since there is no data

available on synaptic time constants we chose the same range for as for the membrane constant. We also set clipping limits for V_{th} , V_0 , V_r such that they are always between the ranges specified in Table 3.2.

There are several ways in which the neuron parameters may be trained. One possibility is to make all neurons in a layer share the same neuron parameters. That is, a single value of V_{th} , V_0 , V_r , α , β is trained and shared by all neurons in a layer. Another possibility is to optimise each of these parameters in each neuron individually as we have done in our experiments. We also always trained the weight matrices $\mathbf{W}^{(l)}$ and $\mathbf{V}^{(l)}$. Training was done by using automatic differentiation on PyTorch [200] and Adam optimiser with learning rate 10^{-3} and betas (0.9, 0.999).

In all surrogate gradient descent experiments a single recurrent spiking layer with 128 neurons received all input spikes. This recurrent layer is followed by a feedforward readout layer with V_{th} set to infinity and with as many neurons as classes in the dataset.

For the loss function, we follow the *max-over-time* loss in [191] to take the maximal membrane potential over the entire time in the readout layer, which was found to lead to the highest classification performance. We then take these potentials and compute the cross-entropy loss

$$\mathcal{L} \left(V_{class}^{(L)}[t] \right) = -\log \left(\frac{\exp(\arg \max_t V_{class}^{(L)}[t])}{\sum_j \exp(\arg \max_t V_j^{(L)}[t])} \right) \quad (3.9)$$

where *class* corresponds to the readout neuron index of the correct label for a given sample. The loss is computed as the average of N_{batch} training samples. This was repeated for a total of N_{epochs} .

In order to improve generalisation we added noise to the input by adding spikes following a Poisson process with rate 1.2 Hz and deleting spikes with probability

0.001.

The parameters used for the network are given in Table 3.2 and Table 3.3 unless otherwise specified. In Fig. 3.2F we used a log-normal distribution for the initial values of the time constants in which we ensured the mode was the same as in the Gamma distribution we used for the other experiments (see Table 3.2) but we scaled the standard deviation to be f times that of the original Gamma distribution. We used $f=4$ for the heterogeneous case.

For the intermediate case all neurons were initialised as in the Homogeneous configuration except for 5% of them selected randomly that were given the largest time constant value allowed of 100ms. The aim of this was to test whether all that was needed was some neurons with a longer memory. We chose 5% since the optimal distribution of trained time constants in the SHD shows that about this fraction of neurons reach this maximum value.

Table 3.2: Parameter initialisation for the different configurations

Parameter	HomInit	HetInit	Description
τ_m	$\bar{\tau}_m$	$\Gamma(3, \bar{\tau}_m/3)$	Membrane time constant
τ_s	$\bar{\tau}_s$	$\Gamma(3, \bar{\tau}_s/3)$	Synaptic time constant
V_{th}	V_{th}	$\mathcal{U}(0.5, 1.5)$	Membrane threshold
V_0	V_0	$\mathcal{U}(-0.5, 0.5)$	Resting potential
V_r	V_r	$\mathcal{U}(-0.5, 0.5)$	Reset potential

Table 3.3: Surrogate Gradient Descent network parameters

Parameter	Value	Description
Δt	0.5 ms	Simulation time step
$\bar{\tau}_m$	20 ms	Mean membrane time constant
$\bar{\tau}_s$	10 ms	Mean synaptic time constant
V_{th}	1V	Membrane threshold
V_0	0 mV	Resting potential
V_r	0 mV	Reset potential
t_{ref}	0 ms	Refractory time
ρ	100	Surrogate steepness

All states $I_j^{(l)}[0]$ and $V_j^{(l)}[0]$ are initialised to 0. For the weights $\mathbf{W}$ and $\mathbf{V}$, we

independently sampled from a uniform distribution $\mathcal{U}(-k^{-1/2}, k^{-1/2})$, with k being the number of afferent connections [201].

3.3.3 FORCE Training

The FORCE method is used to train a network consisting of a single recurrent layer of LIF neurons as in [152]. In particular, we followed the method used to learn to reproduce a songbird singing. In this method, there are no feedforward weights and only the recurrent weights $\mathbf{V}$ are trained. We can express these weights as

$$\mathbf{V} = G\mathbf{v} + Q\boldsymbol{\eta}\boldsymbol{\phi}^T \quad (3.10)$$

The first term in Equation (3.10), namely $G\mathbf{v}$, remains static during training and it is initialised to set the network into chaotic spiking. The learned component of the weights $\boldsymbol{\phi}^T \in \mathbb{R}^{K \times N}$, with N being the number of recurrent units and K the number of encodings, is updated using the Recursive Least Squares algorithm [153]. The vector $\boldsymbol{\eta} \in \mathbb{R}^{N \times K}$ serves as a decoder and it is static during learning. The constants G and Q govern the ratio between chaotic and learned weights.

With this definition of $\mathbf{V}$ we can write the currents into the neurons as the sum $\mathbf{I}[t] = \mathbf{I}_G[t] + \mathbf{I}_Q[t]$ (we dropped the layer l superscript since we only have a single layer) where we define

$$\mathbf{I}_G[t+1] = \alpha\mathbf{I}_G[t] + G\mathbf{v}\mathbf{s}[t] \quad (3.11)$$

$$\mathbf{I}_Q[t+1] = Q\boldsymbol{\eta}\boldsymbol{\phi}^T\mathbf{r}[t] \quad (3.12)$$

$$\mathbf{r}[t+1] = \alpha\mathbf{r}[t] + \mathbf{s}[t] \quad (3.13)$$

In order to stabilise the network dynamics we add a High Dimensional Temporal Signal (HDTS) as in [152]. This is an M dimensional periodic signal $\mathbf{z}[t]$. Given the HDTS period T , we split the interval $[0, T]$ into M subintervals I_m , $m = 1, \dots, M$ such that each of the components of $\mathbf{z}[t]$ is given by

$$z_m[t] = \begin{cases} \left| A \sin \left(\frac{M\pi t}{T} \right) \right|, & \text{if } t \in I_m \\ 0 & \text{otherwise} \end{cases} \quad (3.14)$$

This signal is then projected onto the neurons through a decoder $\boldsymbol{\mu} \in \mathbb{R}^{N \times M}$ similar to $\boldsymbol{\eta}$ in Equation (3.10). The total current is then given by

$$\mathbf{I}[t] = \mathbf{I}_G[t] + \mathbf{I}_Q[t] + \boldsymbol{\mu} \mathbf{z}[t] \quad (3.15)$$

The aim of FORCE learning is to approximate a K -dimensional time varying teaching signal $\mathbf{x}[t]$. The vector $\mathbf{r}[t]$ is used to obtain an approximant of the desired signal

$$\hat{\mathbf{x}}[t] = \boldsymbol{\phi}^T \mathbf{r}[t] \quad (3.16)$$

The weights are updated using the RLS learning rule [153] according to:

$$\boldsymbol{\phi}[t] = \boldsymbol{\phi}[t-1] - \mathbf{e}[t] \mathbf{P}[t] \mathbf{r}[t] \quad (3.17)$$

$$\mathbf{P}[t] = \mathbf{P}[t-1] - \frac{\mathbf{P}[t-1] \mathbf{r}[t] \mathbf{r}[t]^T \mathbf{P}[t-1]}{1 + \mathbf{r}[t]^T \mathbf{P}[t-1] \mathbf{r}[t]} \quad (3.18)$$

During the training phase, the teaching signal $\mathbf{x}[t]$ is used to perform the RLS update. Then, during the testing phase, the teaching signal is removed.

We used the parameters given in Table 3.4 for all FORCE experiments unless otherwise specified. For the gamma initialised synaptic time constants we draw the values from $\Gamma(3, 0.025)$.

Table 3.4: FORCE network parameters

Parameter	Value	Description
Δt	0.04 ms	Simulation time step
N	1000	Number of neurons
M	500	Number of HDTS subintervals
τ_m	10 ms	Membrane time constant
τ_s	20 ms	Synaptic time constant
V_{th}	-40 mV	Membrane threshold
V_0	0 mV	Resting potential
V_r	-65 mV	Reset potential
t_{ref}	2 ms	Refractory time
Q	10	Chaotic weight
G	0.04	Learned weight
A	80	HDTS amplitude

The period T was chosen to be equal to the length of the teaching signal $\mathbf{x}[t]$. The membrane potentials were randomly initialised following a uniform distribution $\mathcal{U}(V_r, V_{th})$. Vectors $\boldsymbol{\eta}$ and $\boldsymbol{\mu}$ were randomly drawn from $\mathcal{U}(-1, 1)$. The static weights $\mathbf{v}$ were drawn from a normal distribution $\mathcal{N}(0, 1/(Np^2))$, then these weights were set to 0 with probability $p = 0.1$. All other variables are initialised to zero unless otherwise specified.

We computed the errors following

$$\log\text{MSE}(\mathbf{x}, \hat{\mathbf{x}}) = \log_{10} \left(\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_F^2}{D} \right) \quad (3.19)$$

Where $\|\mathbf{x}\|_F$ is the Frobenius norm of the multidimensional time signal $\mathbf{x}$ and D is the number of elements in $\mathbf{x}$.

3.4 Discussion and conclusions

We trained spiking neural networks at difficult classification tasks, either forcing all neuron time constants to be the same (homogeneous) or allowing them to be different (heterogeneous). We found that introducing heterogeneity improved the overall performance across a range of tasks and training methods, but particularly so on tasks with richer intrinsic temporal structure. Learning was more robust, for heterogeneous networks, in that the networks were able to learn across a range of different environments, and when the hyperparameters of learning were mistuned. When the learning rule was allowed to tune the time constants as well as synaptic weights, a consistent distribution of time constants was found, akin to a log normal or gamma distribution, and this qualitatively matched time constants measured in experimental data. Our model is consistent with the two possibilities that these time constant distributions are learned during an individuals lifetime, or that they are found as a result of an evolutionary process.

We conclude from this that neural heterogeneity is a metabolically efficient strategy for the brain. Heterogeneous networks have no additional cost in terms of the number of neurons or synapses, and perform as well as homogeneous networks which have an order of magnitude more neurons. This gain also extends to neuromorphic computing systems, as adding heterogeneity to the neuron model adds an additional time and memory cost of only $O(n)$, while adding more neurons has a cost of $O(n^2)$ (because all neurons are connected). Further, in some neuromorphic systems like BrainScaleS this heterogeneity is already present as part of the manufacturing process [202]. In addition to their overall performance being better, heterogeneous networks are more robust and able to learn across a wider range of environments, which is clearly ethologically advantageous. Again, this has a corresponding benefit to neuromorphic computing and potentially machine learning more generally, in that it reduces the cost of hyperparameter tuning, which is often one of the largest

costs for developing these models.

We now turn to the matter of the extent of time constant tuning in real nervous systems. It could be the case that the heterogeneous distribution of time constants observed in different animals and brain regions (Fig. 3.2C) is simply a product of noisy developmental processes. We cannot rule this out. However, our results show that the distributions observed experimentally closely match the optimal ones found by simulations, and this optimal distribution confers a substantial computational advantage. It therefore seems likely that the brain makes use of this advantage. We found that any degree of heterogeneity improves performance, but that the best performance could be found by tuning the distribution of time constants to match the task. Without a more detailed model of particular brain regions in specific animals, and the real world tasks they solve, it is difficult to conclude whether or not the precise distributions observed experimentally are tuned to those tasks or not. Further, having a less precisely tuned distribution may lead to greater robustness in uncertain environments.

A number of studies have used heterogeneous or tunable time constants [203, 204, 205], but these have generally been focused on maximising performance for neuro-morphic applications, and not considering the potential role in real nervous systems. In particular, we have shown that: heterogeneity is particularly important for the type of temporally complex tasks faced in real environments, as compared to the static ones often considered in machine learning; heterogeneity confers robustness allowing for learning in a wide range of environments; optimal distributions of time constants are consistent across training runs and match experimental data; and that our results are not specific to a particular task or training method.

The methods used here are very computationally demanding, and this has limited us to investigating very small networks (hundreds of neurons when using surrogate gradient descent and thousands using FORCE training). Indeed, we estimate that

in the preparation of this paper we used approximately 2 years of GPU computing. Finding new algorithms to allow us to scale these methods to larger networks will be a critical task for the field.

Finally, it would be interesting to see to what extent different forms of heterogeneity confers other advantages, such as spatial as well as temporal heterogeneity. We observed that in the brain, different cell types have different stereotyped distributions of time constants, and it would be interesting to extend our methods to networks with multiple cell types, including more biophysically detailed cell models.

Our computational results show a compelling advantage for heterogeneity, and this makes intuitive sense. Having heterogeneous time constants in a layer allows the network to integrate incoming spikes at different time scales, corresponding to shorter or longer memory traces, thus allowing the readout layer to capture information at several scales and represent a richer set of functions. It would be very valuable to extend this line of thought and find a rigorous theoretical explanation of the advantage of heterogeneity.

Chapter 4

Sparse Spiking Gradient Descent

In this chapter we introduce a new method for computing the backward pass on SNNs that leverages the sparse nature of the spiking activations. We show that we can compute sparse gradients without loss of accuracy and empirically show that we can perform this gradient computation efficiently. We test this on different network architectures and challenging benchmarks resulting in up to a 150x speedup and 85% more memory efficient backward execution. These results open the possibility of a fast, energy and memory efficient end-to-end training on spiking neural networks which benefits neuromorphic training and computational neuroscience research.

4.1 Introduction

In recent years, deep artificial neural networks (ANNs) have matched and occasionally surpassed human-level performance on increasingly difficult auditory and visual recognition problems [1, 2, 3], natural language processing tasks [4, 5] and games [6, 7, 8]. As these tasks become more challenging the neural networks required to solve them grow larger and consequently their power efficiency becomes more important [206, 207]. At the same time, the increasing interest in deploying these models

into embedded applications calls for faster, more efficient and less memory intensive networks [208, 209].

The human brain manages to perform similar and even more complicated tasks while only consuming about 20W [19] which contrasts with the hundreds of watts required for running ANNs [206]. Unlike ANNs, biological neurons in the brain communicate through discrete events called spikes. A biological neuron integrates incoming spikes from other neurons in its membrane potential and after reaching a threshold emits a spike and resets its potential [24]. The spiking neuron, combined with the sparse connectivity of the brain, results in a highly spatio-temporally sparse activity [210] which is fundamental to achieve this level of energy efficiency.

Inspired by the extraordinary performance of the brain, neuromorphic computing aims to obtain the same level of energy efficiency preferably while maintaining an accuracy standard on par with ANNs by emulating spiking neural networks (SNNs) on specialised hardware [211, 212, 146, 132]. These efforts go in two directions: emulating the SNNs and training the SNNs. While there is a growing number of successful neuromorphic implementations for the former [213, 214, 215, 216, 217, 212, 218, 219], the latter has proven to be more challenging. Some neuromorphic chips implement local learning rules [219, 220, 221] and recent advances have achieved to approximate backpropagation on-chip [222]. However, a full end-to-end supervised learning via error backpropagation requires off-chip training [215, 216, 146]. This is usually achieved by simulating and training the entire SNN on a GPU and more recently, emulating the forward pass of the SNN on neuromorphic hardware and then performing the backward pass on a GPU [202, 223]. The great flexibility provided by GPUs allows the development of complex training pipelines without being constrained by neuromorphic hardware specifications. However, as GPUs do not leverage the event-driven nature of spiking computations this results in effective but slower and less energy efficient training. An alternative to directly training

an SNN consists on training an ANN and converting it to an SNN. However, this approach has been recently shown to not be more energy efficient than using the original ANN in the first place [137].

Training SNNs has been a challenge itself even without considering a power budget due to the all-or-none nature of spikes which hinders traditional gradient descent methods [11, 12]. Since the functional form of a spike is a unit impulse, it has a zero derivative for all membrane potentials except at the threshold where it is infinity. Recently developed methods based on surrogate gradients have been shown to be capable of solving complex tasks to near state-of-the-art accuracies [224, 146, 140, 142, 145, 225, 223]. The problem of non-differentiability is solved by adopting a well-behaved surrogate gradient when backpropagating the error [11]. This method, while effective for training very accurate models, is still constrained to working on dense tensors, thus, not profiting from the sparse nature of spiking, and limits the training speed and power efficiency when training SNNs.

In this chapter we introduce a sparse backpropagation method for SNNs. Recent work on ANNs has shown that adaptive dropout [226] and adaptive sparsity [227] can be used to achieve state-of-the-art performance at a fraction of the computational cost [228, 229]. We show that by redefining the surrogate gradient functional form, a sparse learning rule for SNNs arises naturally as a three-factor Hebbian-like learning rule. We empirically show an improved backpropagation time up to 150x faster than current implementations and up to 85% more memory efficient in real datasets (Fashion-MNIST (F-MNIST) [189], Neuromorphic-MNIST (N-MNIST) [188] and Spiking Heidelberg Dataset (SHD) [191]) thus reducing the computational gap between the forward and backward passes. This improvement will not only impact training for neuromorphic computing applications but also for computational neuroscience research involving training on SNNs [225].

4.2 Methods

4.2.1 Forward Model

We begin by introducing the spiking networks we are going to work with (Figs. 4.1 and 4.2). For simplicity we have omitted the batch dimension in the derivations but it is later used on the complexity analysis and experiments.

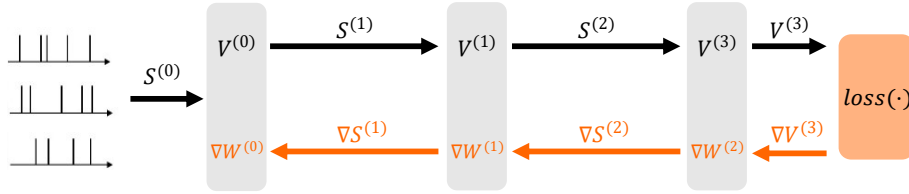


Figure 4.1: Spiking Neural Network diagram highlighting the forward and backward passes

We have a total of L fully connected spiking layers. Each layer l consisting of $n^{(l)}$ spiking neurons which are fully connected to the next layer $l+1$ through synaptic weights $W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l+1)}}$. Each neuron has an internal state variable, the membrane potential, $v_j^{(l)}[t] \in \mathbb{R}$ that updates every discrete time step $t \in \{0, \dots, T-1\}$ for some finite simulation time $T \in \mathbb{N}$. Neurons emit spikes according to a spiking function $f: \mathbb{R} \rightarrow \{0, 1\}$ such that

$$s_i^{(l)}[t] = f(v_i^{(l)}[t]) \quad (4.1)$$

The membrane potential varies according to a simplified Leaky Integrate and Fire (LIF) neuron model in which input spikes are directly integrated into the membrane [24]. After a neuron spikes, its potential is reset to V_r . We will work with a discretised version of this model. The membrane potential of neuron j in layer $l+1$ evolves according to the following difference equation

$$v_j^{(l+1)}[t+1] = \alpha v_j^{(l+1)}[t] + (1 - \alpha)V_0 + \sum_i s_i^{(l)}[t]W_{ij}^{(l)} - (V_{th} - V_r)s_j^{(l+1)}[t] \quad (4.2)$$

The first two terms account for the leaky part of the model where we define $\alpha = \exp(-\Delta t/\tau)$ with $\Delta t, \tau \in \mathbb{R}$ being the time resolution of the simulation and the membrane time constant of the neurons respectively. The second term deals with the integration of spikes. The last term models the resetting mechanism by subtracting the distance from the threshold to the reset potential. For simplicity and without loss of generality, we consider $V_0=0, V_{th}=1, V_r=0$ and $v_j^{(l)}[0]=0 \forall l, j$ from now on.

We can unroll (4.2) to obtain

$$v_j^{(l+1)}[t+1] = \underbrace{\sum_i \underbrace{\sum_{k=0}^t \alpha^{t-k} s_i^{(l)}[k]}_{\text{Input trace}} W_{ij}^{(l)}}_{\text{Weighted input}} - \underbrace{\sum_{k=0}^t \alpha^{t-k} s_j^{(l+1)}[k]}_{\text{Resetting term}} \quad (4.3)$$

Thus, a spiking neural network can be viewed as a special type of recurrent neural network where the activation function of each neuron is the spiking function $f(\cdot)$. The spiking function, is commonly defined as the unit step function centered at a particular threshold V_{th}

$$f(v) = \begin{cases} 1, & v \geq V_{th} \\ 0, & \text{otherwise} \end{cases} \quad (4.4)$$

Note that while this function is easy to compute in the forward pass, its derivative, the unit impulse function, is problematic for backpropagation which has resulted in adopting surrogate derivatives [11] to allow the gradient to flow. Interestingly, it has been shown that surrogate gradient descent performs robustly for a wide range

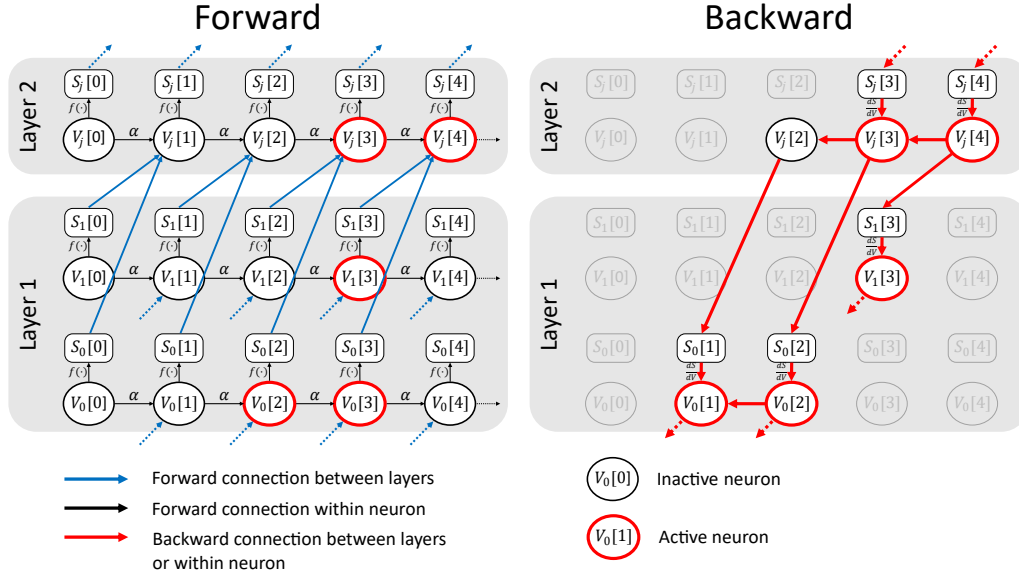


Figure 4.2: Illustration of the gradient backpropagating only through active neurons.

of surrogate functional forms [145].

After the final layer, a loss function $loss(\cdot)$ computes how far the network activity is with respect to some target value for the given input. The loss function can be defined as a function of the network spikes, membrane potentials or both. It may also be evaluated at every single time step or every several steps. We deliberately leave the particular definition of this function open as it does not directly affect the sparse gradient descent method we introduce.

4.2.2 Backward Model

The SNN is updated by computing the gradient of the loss function with respect to the weights in each layer. This can be achieved using backpropagation through time (BPTT) on the unrolled network. Following (4.1) and (4.3) we can derive weight gradients to be

$$\nabla W_{ij}^{(l)} = \sum_t \underbrace{\varepsilon_j^{(l+1)}[t]}_{\text{Gradient from next layer}} \underbrace{\frac{ds_j^{(l+1)}[t]}{dv_j^{(l+1)}[t]}}_{\text{Spike derivative}} \underbrace{\left(\sum_{k < t} \alpha^{t-k-1} s_i^{(l)}[k] \right)}_{\text{Input trace}} \quad (4.5)$$

This gradient consists of a sum over all time steps of the product of three terms. The first term, modulates the spatio-temporal credit assignment of the weights by propagating the gradient from the next layer. The second term, is the derivative of $f(\cdot)$ evaluated at a given $v_j[t]$. The last term is a filtered version of the input spikes. Notice that it is not necessary to compute this last term as this is already computed in the forward pass (4.3). We have chosen not make the spike resetting term in (4.3) differentiable as it has been shown that doing so can result in worse testing accuracy [145] and it increases the computational cost.

For a typical loss function defined on the output layer activity, the value of $\varepsilon_j^{(L-1)}[t]$ (i.e. in the last layer) is the gradient of the loss with respect to the output layer (see Fig. 4.1). For all other layers we have that $\varepsilon_j^{(l)}[t] = \nabla s_j^{(l)}[t]$, that is, the gradient of the loss function with respect to the output spikes of layer l . We can obtain an expression for $\nabla s_j^{(l)}[t]$ as a function of the gradient of the next layer:

$$\begin{aligned} \varepsilon_i^{(l)}[t] = \nabla s_i^{(l)}[t] &= \sum_j W_{ij} \left(\sum_{k > t} \underbrace{\nabla s_j^{(l+1)}[k]}_{\text{Gradient from next layer}} \underbrace{\frac{ds_j^{(l+1)}[k]}{dv_j^{(l+1)}[k]}}_{\text{Spike derivative}} \alpha^{k-t-1} \right) \\ &= \sum_j W_{ij} \delta_j^{(l+1)}[t], \quad l = \{0, \dots, L-2\} \end{aligned} \quad (4.6)$$

All hidden layers need to compute equations (4.5) and (4.6). The first one to update their input synaptic weights and the second one to backpropagate the loss to the previous layer. The first layer ($l=1$) only needs to compute (4.5) and the last layer

($l=L-1$) will compute a different $\varepsilon_j^{(L-1)}[t]$ depending on the loss function.

We introduce the following definitions which result in the sparse backpropagation learning rule.

Definition 4.1. *Given a backpropagation threshold $B_{th} \in \mathbb{R}$ we say neuron j is **active** at time t iff*

$$|v_j[t] - V_{th}| < B_{th} \quad (4.7)$$

Definition 4.2. *The spike gradient is defined as*

$$\frac{ds_j[t]}{dv_j[t]} := \begin{cases} g(v_j[t]), & \text{if } v_j[t] \text{ is active} \\ 0, & \text{otherwise} \end{cases} \quad (4.8)$$

This means that neurons are only active when their potential is close to the threshold. Applying these two definitions to (4.5) and (4.6) results in the gradients only backpropagating through active neurons at each time step as shown in Fig. 4.2. The consequences of this are readily available for the weight gradient in (4.5) as the only terms in the sum that will need to be computed are those in which the postsynaptic neuron j was active at time t . Resulting in the following gradient update:

$$\nabla W_{ij}^{(l)}[t] = \begin{cases} \varepsilon_j^{(l+1)}[t] \frac{ds_j^{(l+1)}[t]}{dv_j^{(l+1)}[t]} \left(\sum_{k < t} \alpha^{t-k-1} s_i^{(l)}[t] \right), & v_j^{(l+1)}[t] \text{ is active,} \\ 0, & \text{otherwise} \end{cases} \quad (4.9)$$

$$\nabla W_{ij}^{(l)} = \sum_t \nabla W_{ij}^{(l)}[t] \quad (4.10)$$

For the spike gradient in (4.6) we have two consequences, firstly, we will only need to compute $\varepsilon_j^{(l)}[t]$ for active neurons since $\nabla s_j^{(l)}[t]$ is always multiplied by $\frac{ds_j^{(l)}[t]}{dv_j^{(l)}[t]}$ in

(4.5) and (4.6). Secondly, we can use a recurrent relation to save time and memory when computing $\delta_j[t]$ in (4.6) as shown in the following proposition.

Proposition 4.1. *We can use a recurrent relation to compute $\delta_j[t]$ given by*

$$\delta_j[t] = \begin{cases} \alpha^n \delta_j[t+n], & \text{if } \frac{ds_j[k]}{dv_j[k]} = 0, \text{ for } t+1 \leq k \leq t+n \\ \nabla s_j[t+1] \frac{ds_j[t+1]}{dv_j[t+1]} + \alpha^n \delta_j[t+n], & \text{if } \frac{ds_j[k]}{dv_j[k]} = 0, \text{ for } t+1 < k \leq t+n, \frac{ds_j[t+1]}{dv_j[t+1]} \neq 0 \end{cases} \quad (4.11)$$

This means that we only need to compute $\delta_j^{(l+1)}[t]$ at those time steps in which either $\frac{ds_j[t+1]}{dv_j[t+1]} \neq 0$ or $v_j^{(l)}[t]$ is active. Thus, we end up with the following sparse expression for computing the spike gradient.

$$\nabla s_j^{(l)}[t] = \begin{cases} \sum_j W_{ij} \delta_j^{(l+1)}[t], & v_j^{(l)}[t] \text{ is active} \\ 0, & \text{otherwise} \end{cases} \quad (4.12)$$

Proof. *The proof follows directly from the definition of $\delta_j[t]$. For the first case we have*

$$\begin{aligned} \delta_j[t] &= \sum_{k>t} \nabla s_j[k] \frac{ds_j[k]}{dv_j[k]} \alpha^{k-t-1} \\ &= \nabla s_j[t+1] \frac{ds_j[t+1]}{dv_j[t+1]} \alpha^0 + s_j[t+2] \frac{ds_j[t+2]}{dv_j[t+2]} \alpha^1 + \dots + \nabla s_j[t+n] \frac{ds_j[t+n]}{dv_j[t+n]} \alpha^{n-1} \\ &\quad + \nabla s_j[t+n+1] \frac{ds_j[t+n+1]}{dv_j[t+n+1]} \alpha^n + \dots \\ &= \nabla s_j[t+1] \frac{ds_j[t+1]}{dv_j[t+1]} \alpha^0 + s_j[t+2] \frac{ds_j[t+2]}{dv_j[t+2]} \alpha^1 + \dots + \nabla s_j[t+n] \frac{ds_j[t+n]}{dv_j[t+n]} \alpha^{n-1} \\ &\quad + \sum_{k>t+n} \nabla s_j[k] \frac{ds_j[k]}{dv_j[k]} \alpha^{k-t-1} \\ &= \alpha^n \sum_{k>t+n} \nabla s_j[k] \frac{ds_j[k]}{dv_j[k]} \alpha^{k-(t+n)-1} \\ &= \alpha^n \delta_j[t+n] \end{aligned}$$

where the penultimate equality holds from the condition $\frac{ds_j[k]}{dv_j[k]} = 0$, for $t+1 \leq k \leq t+n$.

The proof of the second case is identical except that now the term $\nabla s_j[t+1] \frac{ds_j[t+1]}{dv_j[t+1]}$ will be non-zero.

4.2.3 Visualisation of $\delta_j[t]$ computation

We show now a visualisation of the update rule in Proposition 4.1. Given a presynaptic neuron i and postsynaptic neuron j we need to compute the gradient

$$\begin{aligned} \nabla s_i[t] &= \sum_j W_{ij} \left(\sum_{k>t} \nabla s_j[t] \frac{ds_j[t]}{dv_j[t]} \alpha^{k-t-1} \right) \\ &= \sum_j W_{ij} \delta_j[t] \end{aligned} \quad (4.13)$$

We first note that $\nabla s_i[t]$ is always multiplied by $\frac{ds_i[t]}{dv_i[t]}$ either in the weight gradient equation (4.5) or in the spike gradient equation (4.6). Thus, we only need $\nabla s_i[t]$ at those times at which $\frac{ds_i[t]}{dv_i[t]} \neq 0$.

Secondly, according to Proposition 1 $\delta_j[t]$ is only modified beyond an attenuation factor of α^n at those times when $\frac{ds_j[t]}{dv_j[t]} \neq 0$.

We define

$$\begin{aligned} t_i &= \{t : \frac{ds_i[t]}{dv_i[t]} \neq 0\} \quad \text{The times we need to compute the gradient of presynaptic neuron } i \\ t_j &= \{t : \frac{ds_j[t]}{dv_j[t]} \neq 0\} \quad \text{The times that affect } \delta_j \text{ beyond } \alpha^n \text{ attenuation} \end{aligned}$$

Both sets of times are known by the time we compute the gradients since they are simply the times at which neurons i and j were active.

The following example shows how the computation of $\delta_j[t]$ would take place using Proposition 1. We note that we only need to write the result in global memory at times t_i . Consequently, once we have all t_i we can simply stop the computation

t	0	1	2	3	4	5	6	7	8	9
t_j		1		3			6		8	
t_i				3		5	6			

$$\begin{aligned}
\delta_j[7] &= \frac{ds_j[8]}{dv_j[8]} && \text{Compute} \\
\delta_j[6] &= \alpha \delta_j[7] && \text{Compute and write} \\
\delta_j[5] &= \frac{ds_j[6]}{dv_j[6]} + \alpha \delta_j[6] && \text{Compute and write} \\
\delta_j[3] &= \alpha^2 \delta_j[5] && \text{Compute and write} \\
\delta_j[2] &= \frac{ds_j[3]}{dv_j[3]} + \alpha \delta_j[3] && \text{Not required} \\
\delta_j[0] &= \frac{ds_j[1]}{dv_j[1]} + \alpha^2 \delta_j[2] && \text{Not required}
\end{aligned}$$

Thus, in the worse case scenario where sets t_i and t_j are disjoint instead of doing T updates we only do $|t_i| + |t_j|$ updates.

4.2.4 Complexity analysis

We define $\rho_l \in [0, 1]$ to be the probability that a neuron is active in layer l at a given time. Table 4.1 summarises the computational complexity of the gradient computation in terms of number of sums and products. We use here N to refer to the number of neurons in either the input or output layer and B to refer to the batch size. In a slight abuse of notation we include the constants ρ_l as part of the complexity expressions.

This gradient as defined in (4.13) is obtained by computing the matrix product between $W \in \mathbb{R}^{N \times N}$ and $\delta^{(l+1)} \in \mathbb{R}^{N \times T}$ per batch. This gives a total of $O(BN^2T)$ products and sums. In the sparse case we only need to compute a fraction ρ_l of the entries in $\nabla S^{(l)}$ thus a total of $O(\rho_l BN^2T)$ products and sums.

There are two main ways of computing $\delta^{(l+1)}$. The naive way requires doing $O(NT^2)$ products and $O(NT^2)$ sums by computing all values with $k > t$ for a given t every time. Alternatively, we can use memoisation to store all values already computed resulting in just $O(NT)$ products and $O(NT)$ sums per batch. Thus resulting in $O(BNT)$ products and sums. In the sparse case, the worst case scenario is when the active times of layers l and $l+1$ are completely disjoint as we saw in the previous section. Meaning we have to compute $O((\rho_l + \rho_{l+1})BNT^2)$ products and sums without memoisation or $O((\rho_l + \rho_{l+1})BNT)$ with memoisation.

Overall we get that if we do not use memoisation for computing $\nabla S^{(l)}$ then we do $O(B(NT^2 + N^2T))$ sums and products and $O(B((\rho_{l+1} + \rho_l)NT^2 + \rho_{l+1}N^2T))$ in the sparse case. With memoisation we do $O(BN^2T)$ and $O(\rho_{l+1}BN^2T)$ in the sparse case.

Table 4.1: Computational complexity

	Original	Sparse
Sums $\nabla W^{(l)}$	$O(BTN^2)$	$O(\rho_{l+1}BTN^2)$
Products $\nabla W^{(l)}$	$O(BTN^2)$	$O(\rho_{l+1}BTN^2)$
Sums $\nabla S^{(l)}$	$O(BTN^2)$	$O(\rho_l BTN^2)$
Products $\nabla S^{(l)}$	$O(BTN^2)$	$O(\rho_l BTN^2)$

In order for the sparse gradients to work better than their dense counterpart we need to have a balance between having few enough active neurons as to make the sparse backpropagation efficient while at the same time keeping enough neurons active to allow the gradient to backpropagate. We later show in section 4.3 that this is the case when testing it on real world datasets.

4.2.5 Sparse learning rule interpretation

The surrogate derivative we propose, while not having a constrained functional form for active neurons, imposes a zero gradient for the inactive ones. Looking back into equation (4.5) we can identify it now as a three-factor Hebbian learning rule [230]. With our surrogate derivative definition, the second term (spike derivative) measures whether the postsynaptic neuron is close to spiking (i.e. its membrane potential is close to the threshold). The third term (input trace) measures the presynaptic neuron activity in the most recent timesteps. Finally the first term (gradient from next layer) decides whether the synaptic activity improves or hinders performance with respect to some loss function. We note that this is not exactly a Hebbian learning rule as neurons that get close to spike but do not spike can influence the synaptic weight update and as such, the second term is simply a surrogate of the postsynaptic spike activity.

Previous surrogate gradient descent methods used a similar approach to implement backpropagation on SNNs [146, 140, 205, 225, 145], often using surrogate function definitions that are maximal when the membrane is at the threshold and decay as the potential moves away from it. In fact, in [146] the surrogate function used implies that neurons backpropagate a zero gradient 32% of the time. However, as shown in the next section, neurons can be active a lot less often when presented with spikes generated from real world data and still give the same test accuracy. This is only possible because surrogate gradient approximations to the threshold function yield much larger values when the potential is closer to spike thus concentrating most of the gradient on active neurons. This fact makes it possible to fully profit from our sparse learning rule for a much reduced computational and memory cost.

4.3 Results

We evaluated the correctness and improved efficiency of the proposed sparse back-propagation algorithm on real data of varying difficulty and increasing spatio-temporal complexity. Firstly, the Fashion-MNIST dataset (F-MNIST) [189] is an image dataset that has been previously used on SNNs by converting each pixel analogue value to spike latencies such that a higher intensity results in an earlier spike [225][145]. Importantly, each input neuron can only spike at most once thus resulting in a simple spatio-temporal complexity and very sparse coding. Secondly, we used the Neuromorphic-MNIST (N-MNIST) [188] dataset where spikes are generated by presenting images of the MNIST dataset to a neuromorphic vision sensor. This dataset presents a higher temporal complexity and lower sparsity than the F-MNIST as each neuron can spike several times and there is noise from the recording device. Finally, we worked on the Spiking Heidelberg Dataset (SHD) [191]). This is a highly complex dataset that was generated by converting spoken digits into spike times by using a detailed model of auditory bushy cells in the cochlear nucleus. This results in a very high spatio-temporal complexity and lowest input sparsity. A visualisation of samples of each dataset can be found in the supplementary materials. These datasets cover the most relevant applications of neuromorphic systems. Namely, artificially encoding a dataset into spikes (Fashion-MNIST), reading spikes from a neuromorphic sensor such as a DVS camera (N-MNIST) and encoding complex temporally-varying data into spikes (SHD).

We run all experiments on three-layer fully connected network as in Fig. 4.1, where the last layer (readout) has an infinite threshold. Both spiking layers in the middle have the same number of neurons. We used $g(v) := 1/(\beta|v - V_{th}| + 1)^2$ for the surrogate gradient as in [145]. See Appendix B.1 for all training details. We implemented the sparse gradient computation as a PyTorch CUDA extension [200].

4.3.1 Spiking neurons are inactive most of the time resulting in higher energy efficiency

One of the most determining factors of the success of our sparse gradient descent resides on the level of sparsity that we can expect and, consequently, the proportion of active neurons we have in each layer. We measure the percentage of active neurons on each batch as training progresses for each dataset. This can be computed as

$$\text{Activity} = 100 \times \frac{\# \text{ of active neurons}}{BTN} \quad (4.14)$$

where B is the batch size T is the total number of time steps and N the number of neurons in the layer and the number of active neurons is obtained according to (4.7). This is an empirical way of measuring the coefficient ρ_l introduced earlier.

Figure 4.3A shows the activity on each dataset. As expected, the level of activity is correlated with the sparseness of the inputs with the lowest one being on the F-MNIST and the larger in the SHD dataset. Remarkably however, the activity on average is never above 2% on average in any dataset and it is as low as 1.06% in the F-MNIST dataset. This means that on average we will never have to compute more than 2% of the factors in ∇W and 2% of the values of ∇S . This can be visualised in the bottom row of Fig. 4.3B.

These coefficients give us a theoretical upper bound to the amount of energy that can be saved if this sparse backpropagation rule was implemented in specialised hardware which only performed the required sums and products. This is summarised in Table 4.2.

Importantly, in dense backpropagation methods the temporal dimension of datasets and models is necessarily limited to a maximum of ~ 2000 since gradients must be computed at all steps [191, 225]. This means that a lot of negligible gradients are

Table 4.2: Theoretical upper bound to energy saved in hidden layers.

	Mean activity layer 1 (%)	Mean activity layer 2 (%)	Energy saved in ∇W (%)	Energy saved in ∇S (%)
F-MNIST	1.06	0.87	99.13	98.94
N-MNIST	1.12	0.77	99.23	98.88
SHD	1.70	1.09	98.91	98.30

computed. However, our method adapts to the level of activity of the network and only computes non-negligible gradients. This cost-prohibitive reality was evidenced when we attempted to run these experiments in smaller GPUs leading to running out of memory when using dense methods but not on ours (see Fig. 4.6). Thus, sparse spiking backpropagation allows to train data that runs for longer periods of time without requiring a more powerful hardware.

4.3.2 Sparse backpropagation training approximates the gradient very well

We test that our sparse learning rule performs on par with the standard surrogate gradient descent. Figure 4.3B shows a visualisation of the weight (first row) and spike gradients (second row). Note how most of the values of $\nabla S^{(1)}$ were not computed and yet $\nabla W^{(1)}$ is nearly identical to the original gradient. This is further confirmed in Figures 4.3C and 4.3D where the loss on the SHD dataset is practically identical with both methods (the loss for the other datasets can be found in Appendix B.2) and the test accuracy is practically identical (albeit slightly better) in the sparse case (F-MNIST: 82.2%, N-MNIST: 92.7%, SHD: 77.5%). These accuracies are on par to those obtained with similar networks on these datasets (F-MNIST: 80.1%, N-MNIST: 97.4%, SHD: 71.7%) [191, 145, 225]. These results show that sparse spiking backpropagation gives practically identical results to the original backpropagation but at much lower computational cost.

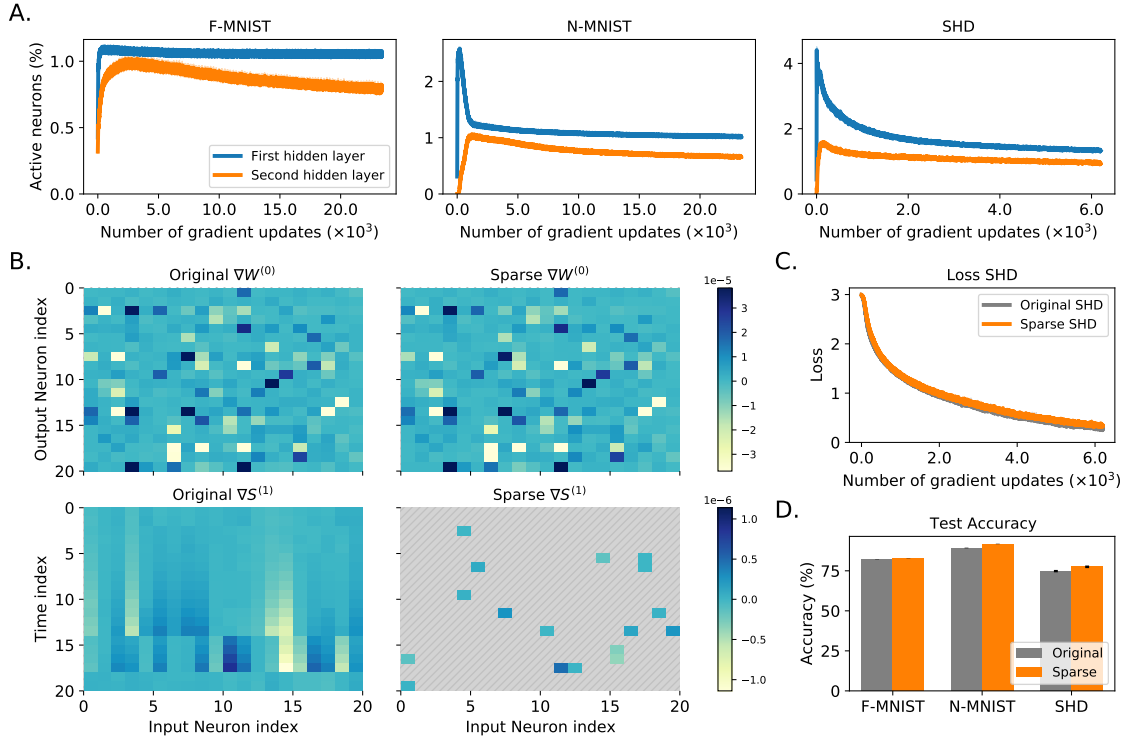


Figure 4.3: Sparse backpropagation learns at high levels of sparsity. All figures except B are a 5 sample average. Standard error in the mean is displayed (although too small to be easily visualised). **A.** Percentage of active neurons during training as a fraction of total tensor size ($B \times T \times N$) with $N = 200$ neurons for each dataset. **B.** Visualisation of the weight and spike gradients on the SHD dataset. We show zero value in hatched grey. Note how both $\nabla W^{(0)}$ are nearly identical in despite being computed using a small fraction of the values of $\nabla s^{(1)}$ in the sparse case. **C.** Loss evolution on the SHD dataset using both algorithms. **D.** Final test accuracy on all datasets using both methods.

4.3.3 Sparse training is faster and less memory intensive

We now measure the time it takes the forward and the backward propagation of the second layer of the network during training as well as the peak GPU memory used. We choose this layer because it needs to compute both $\nabla W^{(1)}$ and $\nabla s^{(1)}$ (note that the first layer only needs to compute $\nabla W^{(0)}$). A performance improvement in this layer translates to an improvement in all subsequent layers in a deeper network. We compute the backward speedup as the ratio of the original backpropagation time and the sparse backpropagation time. We compute the memory saved as

$$\text{Memory saved} = 100 \times \frac{\text{memory_original} - \text{memory_sparse}}{\text{memory_original}} \quad (4.15)$$

Figure 4.5A shows the speedup and memory saved on a 200 neuron hidden layer (both hidden layers). Backpropagation time is faster by 40x and GPU memory is reduced by 35%. To put this speedup into perspective Figure 4.5B shows the time taken for backpropagating the second layer in the original case (about 640ms) and the sparse case (about 17ms). It also shows that the time taken for backpropagating is nearly constant during training as it could be expected after studying the network activity in Fig. 4.3A.

We also tested how robust is this implementation for a varying number of hidden neurons (in both layers). We increase the number of neurons up to 1000 which is similar to the maximum number of hidden units used in previous work [191] and it is also the point at which the original backpropagation method starts to run out of memory. Here we also show the overall speedup defined as the ratio between the sum of forward and backward times of the original over the sparse backpropagation. The results shown in Fig. 4.5C show that the backward speedup and memory saved remain constant as we increase the number of hidden neurons but the overall speedup increases substantially. Given that the forward time is the same for both methods (see Fig. 4.4) this shows that as we increase the number of neurons, the backward time takes most of the computation and thus the sparse backpropagation becomes more important.

We also note that these results vary depending on the GPU used, Figure 4.5 was obtained from running on an RTX6000 GPU. We also run this on smaller GPUs (GTX1060 and GTX1080Ti) and found that the improvement is even better reaching up to a 70x faster backward pass on N-MNIST and SHD. This is shown in Fig: 4.6. Our results show that our method speeds up backward execution between one and

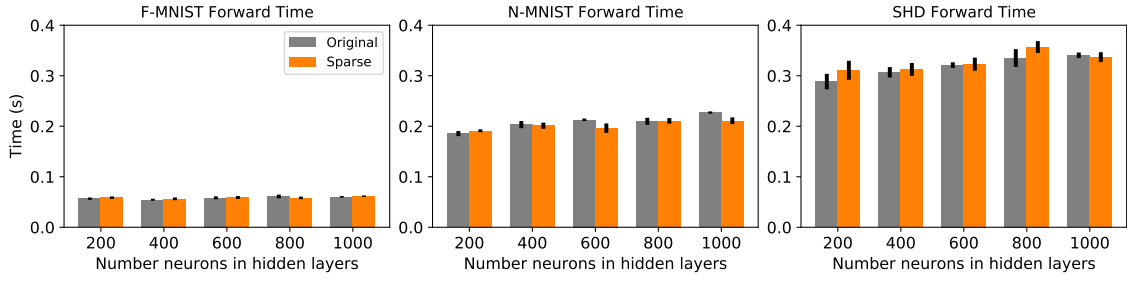


Figure 4.4: Average forward time for each dataset and varying number of neurons (5 samples)

two orders of magnitude, meaning that to a first approximation we would expect it reduce energy usage by the same factor since GPUs continue to use 30-50% of their peak power even when idle [231].

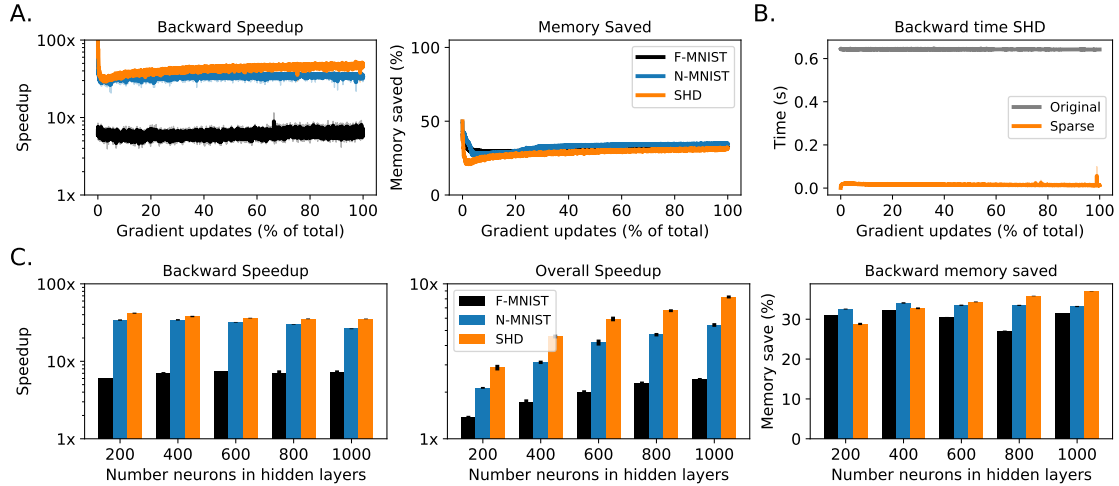


Figure 4.5: Speedup and memory improvement when using sparse gradient back-propagation. All figures are a 5 sample average and its standard error in the mean is also displayed. **A.** Backward speedup and memory saved in the second hidden layer for all three datasets when using sparse gradient descent with 200 hidden neurons. **B.** Time spent in computing the backward pass in the second hidden layer consisting of 200 neurons when using regular gradient descent and sparse gradient descent. **C.** Backward speed and memory saved in the second hidden layer as we increase the number of hidden neurons in all hidden layers. We included the overall speedup which takes into consideration both the forward and backward times spent in the second layer.

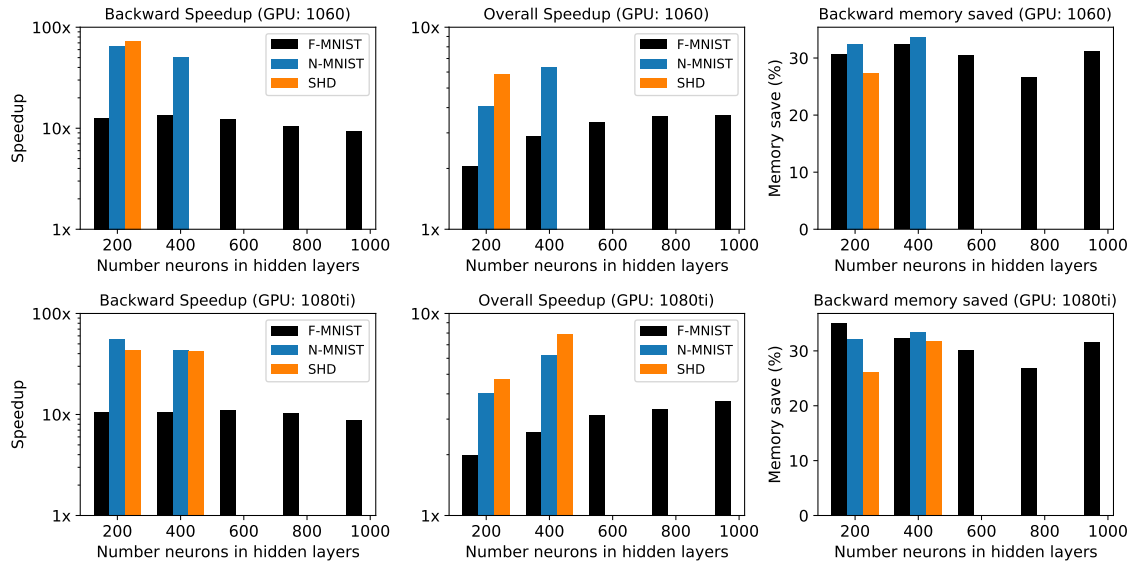


Figure 4.6: Backward speedup, overall layer speedup and backward memory memory saved on an NVIDIA GeForce GTX 1060 (top row) and 1080Ti (bottom row)

4.3.4 Training deeper and more complex architectures

We show that our gradient approximation can successfully train deeper networks by training a 6-layer fully connected network (5 hidden plus readout all with 300 neurons) on Fashion-MNIST (see Fig. 4.7). We measured the performance of the 5 hidden layers and achieve a 15x backward speedup when using 1080-Ti GPU (better than the 10x speedup under these conditions with only 2 hidden layers as shown in Fig. 4.6) and 25% memory saved while achieving a 82.7% testing accuracy (slightly better than with 2 layers at 81.5%). The results are consistent with our previous findings and show that the gradient is not lost in deeper architectures.

We also run a experiment with a convolutional and pooling layers on the Fashion-MNIST dataset obtaining a test accuracy of 86.7% when using our spike gradient approximation and 86.9% with the original gradients. We do not report speedup or memory improvements in this network as developing a sparse convolutional CUDA kernel is out of the scope of our work but we simply ran a dense implementation with clamped gradients. This proves that our gradient approximation is able to

train on more complex architectures although it still remains to be shown whether efficient sparse operators can be implemented for this purpose. See Appendix B.1 for training details.

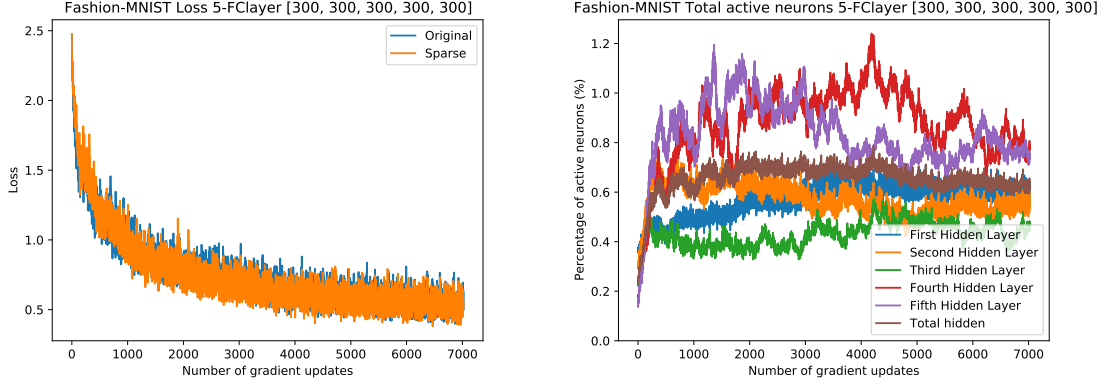


Figure 4.7: Loss history and layer activities when training on Fashion-MNIST with 5 hidden layers with 300 neurons each

4.3.5 Sparsity-Accuracy trade-off

We trained the SHD dataset on the original 2-layer fully connected network with 400 neurons in each hidden layer and varied the backpropagation threshold B_{th} . We found that the training is very robust to even extreme values of B_{th} . The loss and accuracy remains unchanged even when setting $B_{th} = 0.95$ (see 4.8A and 4.8B) achieving a nearly 150x speedup and 85% memory improvement as shown in 4.8C. We also inspected the levels of activity while varying B_{th} and we found that there are enough active neuron to propagate the gradient effectively. As B_{th} gets closer to 1 the number of active neurons decreases rapidly until there are no gradients to propagate.

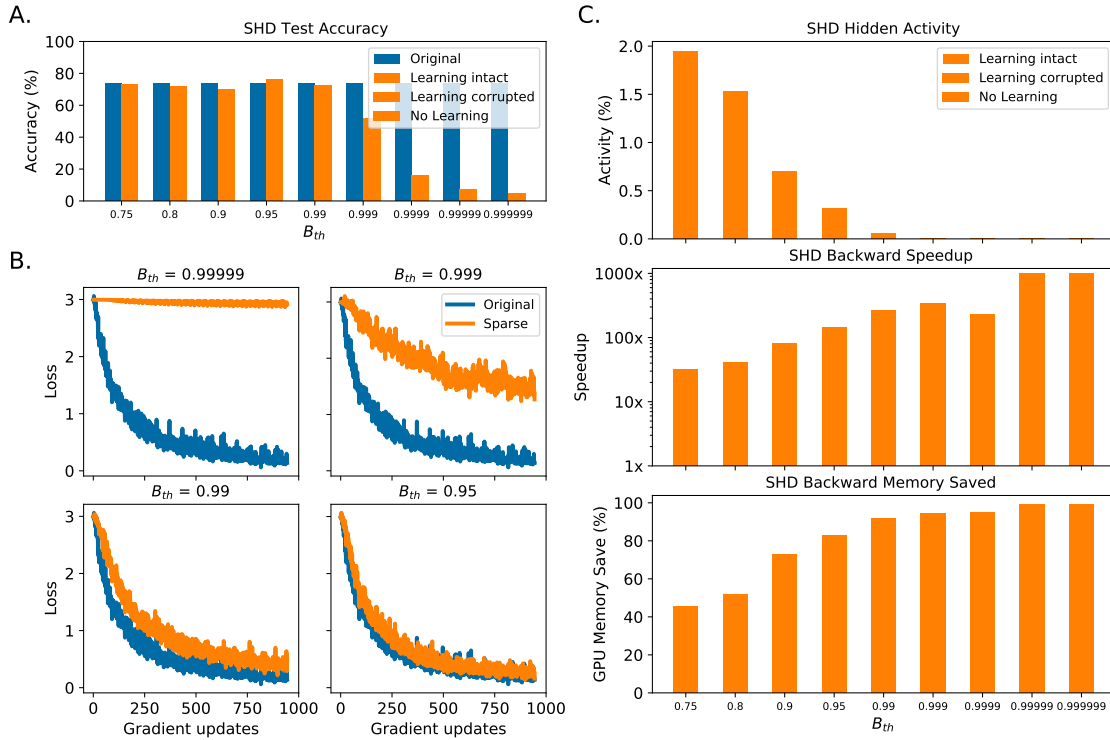


Figure 4.8: Impact of varying the backpropagation threshold B_{th} on the SHD dataset. **A.** Final test accuracy on the SHD dataset as B_{th} increases. **B.** Loss evolution on the SHD dataset for different B_{th} . **C.** Hidden activity, backward speedup and backward memory saved as B_{th} increases.

4.4 Discussion and Conclusions

Recent interest in neuromorphic computing has lead to energy efficient emulation of SNNs in specialised hardware. Most of these systems only support working with fixed network parameters thus requiring an external processor to train. Efforts to include training within the neuromorphic system usually rely on local learning rules which do not guarantee state-of-the-art performance. Thus, training on these systems requires error backpropagation on von-Neumann processors instead of taking advantage of the physics of the neuromorphic substrate [232, 233].

We have developed a sparse backpropagation algorithm for SNNs that achieves the same performance as standard dense backpropagation at a much reduced computational and memory cost. To the best of our knowledge this is the first backpropa-

gation method that leverages the sparse nature of spikes in training and tackles the main performance bottleneck in neuromorphic training. Moreover, while we have used dense all-to-all connectivity between layers, our algorithm is compatible with other SNNs techniques aimed at sparser connectivity like synaptic pruning [234].

By constraining backpropagation to active neurons exclusively, we reduce the number of nodes and edges in the backward computation graph to a point where over 98% of the operations required to compute the gradients can be skipped. This results in a much faster backpropagation time, lower memory requirement and less energy consumption on GPU hardware while not affecting the testing accuracy. This is possible because the surrogate gradient used to approximate the derivative of the spiking function is larger when the membrane potential is closer to the threshold resulting in most of the gradient being concentrated on active neurons. Previous work has used, to a lesser extent, similar ideas. Esser et al. [146] used a surrogate gradient function which effectively resulted in about one third the neurons backpropagating a zero gradient. Later, Pozzi, et al. [235] developed a learning rule where only the weights of neurons directly affecting the activity of the selected output class are updated, however, the trial and error nature of the learning procedure resulted in up to 3.5x slower training.

However, due to the lack of efficient sparse operators in most auto-differentiation platforms, every layer type requires to develop its own sparse CUDA kernel in order to be competitive with current heavily optimised libraries. In our work, we concentrated on implementing and testing this in spiking fully connected layers and we managed to show that sparse backpropagation is a lot faster and uses less memory. Developing these sparse kernels is not trivial and we are aware that having to do so for each layer is an important limitation of our work. This is no surprise since for now there is few reasons to believe that sparse tensor operations will significantly accelerate neural network performance. Our work aims to challenge this view and

motivate the adoption of efficient sparse routines by showing for the first time that sparse BPTT can be faster without losing accuracy.

We implemented and tested our method on real data on fully connected SNNs and simulated it in convolutional SNNs. The same method can be extended to any SNN topology as long as the neurons use a threshold activation function and the input is spiking data. Yet, there remains still the need to develop custom sparse CUDA kernels for each layer type. Developing these kernel types is not trivial and we are aware that having to do so for each layer is an important limitation of our work. This is no surprise since for now there is few reasons to believe that sparse tensor operations will significantly accelerate neural network performance. Our work aims to challenge this view and motivate the adoption of efficient sparse routines by showing for the first time that sparse spiking BPTT can be faster while achieving the same accuracy as its dense counterpart. Nevertheless, ideally, a dynamic backward graph should be generated each time the backward pass needs to be computed and consequently a GPU may not be the best processing unit for this task. A specialised neuromorphic implementation or an FPGA may be more suited to carry out the gradient computation task as this graph changes every batch update. Additionally, while we have reduced the number of operations required for training several times, we have only achieved up to a 85% memory reduction thus, memory requirements remains an important bottleneck.

In summary, our sparse backpropagation algorithm for spiking neural networks tackles the main bottleneck in training SNNs and opens the possibility of a fast, energy and memory efficient end-to-end training on spiking neural networks which will benefit neuromorphic training and computational neuroscience research. Sparse spike backpropagation is a necessary step towards a fully efficient on-chip training which will be ultimately required to achieve the full potential of neuromorphic computing.

Chapter 5

Spiking Network Initialisation and Firing Rate Collapse

5.1 Introduction

The remarkable success of deep artificial neural networks (ANNs) [236, 1] has prompted a new interest in training spiking neural networks (SNNs) with the aim of studying information processing in the brain and advancing neuromorphic engineering [237, 238]. ANNs are typically trained using a gradient descent algorithm that relies on the differentiability of the loss function with respect to the network parameters. However, due to the binary nature of spiking neurons activations, this assumption does not hold true in SNNs. Different methods have been developed to circumvent this issue, one of the most popular being surrogate gradient descent [11, 145]. Surrogate gradient algorithms use a smooth function to approximate the spiking function (Heaviside step function) in the gradient computation. This means that the network still propagates spikes in the forward pass but backpropagates continuous values. Since there are many possible choices of the surrogate function the surrogate gradient is, unlike the true gradient, not unique. Yet, SNNs have been

shown to learn very reliably under a wide variety of surrogate choices provided the network is properly initialised [145].

Similarly to ANNs, a suboptimal weight initialisation on SNNs may result in vanishing or exploding gradients [239, 240]. This issue has been largely explored in the ANN literature and a wide range of strategies has been developed for an optimal parameter initialisation [241]. The most popular methods rely on variance flow analysis and aim to keep constant activations and gradient variance across layers. Depending on the activation function and weight distribution different initialisation schemes have been devised [242, 243, 244, 245]. Other methods have also been developed based on mean-field theory [246], Lipschitz property [247] and Hessian approximations [241] (see [241] for a review).

Spiking Neural Networks are currently initialised following schemes designed for ANNs. These schemes are inadequate as the SNN activation function is different from the ones studied in the ANN literature and, unlike SNNs, ANNs do not have any temporal dynamics or resetting. Typically, this leads to either having too few spikes in the final layers (vanishing activity) or having saturated neurons that fire synchronously at their maximum rate and cannot propagate any information (saturated/exploding activity). As a consequence, we often end up having to manually adjust the weight variance and need to use regularisation penalties to ensure appropriate rates. This problem becomes worse when considering deeper networks where potentially each layer needs to be tuned separately as the spiking signals get lost or gradients vanish.

Research on optimal initialisation schemes on SNNs are very scarce. To the best of our knowledge, this has only been addressed in [240]. However, this work presents two important limitations. Firstly, it only considers approximate membrane distributions that do not include the spiking and resetting mechanisms which ultimately limit the applicability of this method to deeper networks. Secondly, it concentrates

most of the analysis on the forward pass and does not derive analytical results for the backward pass. In this chapter, we aim to extend this work by considering:

- **Variance propagation in forward and backward pass.** We use methods developed for ANNs on SNNs forward and backward propagation. This leads to obtaining a weak bound for the firing rate in the forward pass and a closed form expression for the optimal weight variance in the backward pass.
- **Fokker-Planck and Shot-noise approach for the forward pass.** Since variance propagation is not enough to establish the firing rate in the final layer we resort to the diffusion approximation and similar approaches to compute the output firing rate.
- **Firing rate collapse solution.** As we increase the timestep to typical step sizes used to train SNNs with surrogate gradient methods, the diffusion approximation becomes less accurate. We introduce three different solutions to this problem under different assumptions.

5.2 Background Theory

In this section we introduce the background theory necessary for understanding the remaining of the chapter. This includes forward and backward variance propagation on ANNs for optimal weight initialisation and different methods to compute the output firing rate and membrane distribution of a layer of LIF neurons including: diffusion approximation, shot-noise and threshold integration.

5.2.1 Initialisation based on variance flow in ANNs

The derivations in this section are a more detailed version of those in [243]. We first compute the variance of the activations and then the variance of the gradients on feedforward/convolutional ANNs. From this we derive the optimal weight variance to keep the activations variance bounded. We explore the use of this method on SNNs in section 5.4.

We often use the following identities for random variables X_i and scalars $a_i \in \mathbb{R}$:

$$Var \left[\sum_{i=1}^N a_i X_i \right] = \sum_{i=1}^N a_i^2 Var[X_i] + 2 \sum_{1 \leq i < j \leq N} a_i a_j Cov[X_i, X_j] \quad (5.1)$$

for the special case where X_i, X_j are uncorrelated $\forall i, j$ with $i \neq j$

$$Var \left[\sum_{i=1}^N a_i X_i \right] = \sum_{i=1}^N a_i^2 Var[X_i] \quad (5.2)$$

For independent random variables variables X, Y

$$Var[XY] = E[X^2]E[Y^2] - (E[X])^2 (E[Y])^2 \quad (5.3)$$

Forward variance propagation on ANNs

Unlike SNNs, ANNs do not have any dynamics or resetting present, they simply consist of an affine transformation followed by a non-linearity.

$$y^{(l+1)} = W^{(l)} x^{(l)} + b^{(l)} \quad (5.4)$$

$$x^{(l+1)} = f(y^{(l+1)}) \quad (5.5)$$

where $y^{(l+1)}, x^{(l+1)} \in \mathbb{R}^{n^{(l+1)}}$, $W \in \mathbb{R}^{n^{(l+1)} \times n^{(l)}}$, $x^{(l)} \in \mathbb{R}^{n^{(l)}}$, $b^{(l)} = \mathbf{0}$ at initialisation, f is a non-linear element-wise operation and $l = 0, \dots, L-1$. We make the assumptions

that $x_i^{(l)}$ are i.i.d. for all i and l , $W_{ij}^{(l)}$ and $x_i^{(l)}$ are independent and $W_{ij}^{(l)}$ is zero mean and symmetric around zero and i.i.d for all i, j and l . Then

$$\begin{aligned} \text{Var} \left[y_j^{(l+1)} \right] &= \text{Var} \left[\sum_i W_{ij}^{(l)} x_i^{(l)} + b_j^{(l)} \right] \\ &= n^{(l)} \text{Var} \left[W_{ij}^{(l)} x_i^{(l)} \right] \\ &= n^{(l)} \text{Var} \left[W_{ij}^{(l)} \right] E \left[\left(x_i^{(l)} \right)^2 \right] \end{aligned} \quad (5.6)$$

Given that $W_{ij}^{(l)}$ is zero mean and symmetric so is $y_i^{(l+1)}$. Thus, for a typical case of a ReLU function $f(\cdot) = \max(0, \cdot)$ we have

$$\begin{aligned} E \left[\left(x_i^{(l)} \right)^2 \right] &= E \left[\left(\max(0, y_i^{(l)}) \right)^2 \right] \\ &= \frac{1}{2} E \left[\left(y_i^{(l)} \right)^2 \right] \\ &= \frac{1}{2} \text{Var} \left[y_i^{(l)} \right] \end{aligned} \quad (5.7)$$

Resulting in

$$\text{Var} \left[y_j^{(l+1)} \right] = \frac{1}{2} n^{(l)} \text{Var} \left[W_{ij}^{(l)} \right] \text{Var} \left[y_i^{(l)} \right] \quad (5.8)$$

and for L layers

$$\text{Var} \left[y_j^{(L)} \right] = \text{Var} \left[y_i^{(1)} \right] \prod_{l=1}^{L-1} \frac{1}{2} n^{(l)} \text{Var} \left[W_{ij}^{(l)} \right] \quad (5.9)$$

Ensuring that the variance of the activations does not increase or decrease exponentially with the number of layers, results in Kaiming's initialisation [\[243\]](#)

$$\text{Var} \left[W_{ij}^{(l)} \right] = 2/n^{(l)} \quad (5.10)$$

Backward variance propagation in ANNs

Ideally gradients should propagate from the last layer to the first without exploding or vanishing. To achieve this we follow the same approach as before but now with the gradients.

We begin by rewriting the forward pass equations

$$y^{(l+1)} = W^{(l)}x^{(l)} + b^{(l)} \quad (5.11)$$

$$x^{(l+1)} = f(y^{(l+1)}) \quad (5.12)$$

Where $y^{(l+1)}, x^{(l+1)} \in \mathbb{R}^{n^{(l+1)}}$, $W^{(l)} \in \mathbb{R}^{n^{(l+1)} \times n^{(l)}}$, $x^{(l)} \in \mathbb{R}^{n^{(l)}}$, $b^{(l)} = \mathbf{0}$ at initialisation and f is a non-linear element-wise operation and $l = 0, \dots, L-1$.

Given a loss function $\varepsilon : \mathbb{R}^{n^{(L-1)}} \rightarrow \mathbb{R}$ we can write the gradient of the loss with respect to the activations

$$\frac{\partial \varepsilon}{\partial x^{(l)}} = \frac{\partial \varepsilon}{\partial y^{(l+1)}} \frac{\partial y^{(l+1)}}{\partial x^{(l)}} \quad (5.13)$$

where $\frac{\partial \varepsilon}{\partial x^{(l)}} \in \mathbb{R}^{1 \times n^{(l)}}$, $\frac{\partial \varepsilon}{\partial y^{(l+1)}} \in \mathbb{R}^{1 \times n^{(l+1)}}$ and $\frac{\partial y^{(l+1)}}{\partial x^{(l)}} \in \mathbb{R}^{n^{(l+1)} \times n^{(l)}}$ (note that we are using numerator layout convention). Using (5.12) we can expand

$$\left(\frac{\partial \varepsilon}{\partial y^{(l+1)}} \right)_i = \left(\frac{\partial \varepsilon}{\partial x^{(l+1)}} \right)_i f' \left(y_i^{(l+1)} \right) \quad (5.14)$$

We also note that $\frac{\partial y^{(l+1)}}{\partial x^{(l)}} = \frac{\partial W^{(l)}x^{(l)}}{\partial x^{(l)}} = W^{(l)}$ ¹.

¹While we are presenting the case of the fully connected layer, the following results also hold for other architectures such as convolutional networks by simply adjusting the backward weight matrix $\frac{\partial y^{(l+1)}}{\partial x^{(l)}} = \widehat{W}^{(l)}$. For instance in a regular convolutional layer in the forward pass, we have a filter of size $k \times k$ and a total of c input channels and d output channels: $x^{(l)} \in \mathbb{R}^{c k^2 \times 1}$ and $y^{(l+1)} \in \mathbb{R}^{d \times 1}$ and $W^{(l)} \in \mathbb{R}^{d \times c k^2}$. I.e., $x^{(l)}$ is now an input block (a window spanning all input depths) which has been flattened, $y^{(l+1)}$ is an output block (a single neuron per output depth) and

For notational simplicity, throughout this subsection we use $\frac{\partial \varepsilon}{\partial z} := \Delta z$ for any partial derivative of the loss with respect to z as in [243].

This means we end up with the following equations

$$\Delta x^{(l)} = \Delta y^{(l+1)} W^{(l)} \quad (5.15)$$

$$\Delta y_i^{(l+1)} = \Delta x_i^{(l+1)} f' \left(y_i^{(l+1)} \right) \quad (5.16)$$

We make the assumptions that $\Delta y_i^{(l+1)}$ are i.i.d. for all i and l , $W_{ij}^{(l)}$ and $\Delta y_i^{(l+1)}$ are independent and $W_{ij}^{(l)}$ is zero mean and symmetric around zero and i.i.d for all i, j and l . We further assume that $\Delta x_i^{(l+1)}$ and $f' \left(y_i^{(l+1)} \right)$ are independent.

We want to compute the variance of the gradients of a neuron on a given layer $\Delta x_j^{(l)}$ as a function of the variance of the next layer $\Delta x_i^{(l+1)}$

$$\begin{aligned} Var \left[\Delta x_j^{(l)} \right] &= Var \left[\sum_i W_{ij}^{(l)} \Delta y_i^{(l+1)} \right] \\ &= \sum_i Var \left[W_{ij}^{(l)} \Delta y_i^{(l+1)} \right] \\ &= n^{(l+1)} Var \left[W_{ij}^{(l)} \Delta y_i^{(l+1)} \right] \\ &= n^{(l+1)} \left(E \left[W_{ij}^{(l)2} \right] E \left[\Delta y_i^{(l+1)2} \right] - E \left[W_{ij}^{(l)} \right]^2 E \left[\Delta y_i^{(l+1)} \right]^2 \right) \\ &= n^{(l+1)} Var \left[W_{ij}^{(l)2} \right] E \left[\Delta y_i^{(l+1)2} \right] \end{aligned} \quad (5.17)$$

$W^{(l)}$ is the d filters flattened and stacked. This operation is repeated for each input block. Thus, a neuron in the input affects a total of dk^2 neurons in the output (it helps to think of a single input arriving at a given output neuron by a moving window k times in both height and width). This leads to us defining a new $\hat{x}^{(l)} \in \mathbb{R}^{c \times 1}$, $\hat{y}^{(l+1)} \in \mathbb{R}^{dk^2 \times 1}$ and $\widehat{W}^{(l)} \in \mathbb{R}^{k^2 d \times c}$ and $W^{(l)}$ can be reshaped into $\widehat{W}^{(l)}$. Then the only difference in the results of this subsection are obtained by using $\hat{n}^{(l)} = dk^2$ instead of $n^{(l)} = ck^2$.

We note that

$$\begin{aligned}
 E \left[\Delta y_i^{(l+1)} \right] &= E \left[\Delta x_i^{(l+1)} f' \left(y_i^{(l+1)} \right) \right] \\
 &= E \left[\Delta x_i^{(l+1)} \right] E \left[f' \left(y_i^{(l+1)} \right) \right] \\
 &= E \left[\Delta y_i^{(l+2)} \right] E \left[W_{ij}^{(l+1)} \right] E \left[f' \left(y_i^{(l+1)} \right) \right] \\
 &= 0
 \end{aligned}$$

Since $E \left[W_{ij}^{(l+1)} \right] = 0$ and we assumed that $\Delta x_i^{(l+1)}$ is independent of $f' \left(y_i^{(l+1)} \right)$.

Then we have

$$\begin{aligned}
 E \left[\Delta y_i^{(l+1)2} \right] &= Var \left[\Delta y_i^{(l+1)} \right] \\
 &= Var \left[\Delta x_i^{(l+1)} f' \left(y_i^{(l+1)} \right) \right] \\
 &= E \left[\Delta x_i^{(l+1)2} \right] E \left[f' \left(y_i^{(l+1)} \right)^2 \right] - E \left[\Delta x_i^{(l+1)} \right] E \left[f' \left(y_i^{(l+1)} \right) \right] \\
 &= Var \left[\Delta x_i^{(l+1)} \right] E \left[f' \left(y_i^{(l+1)} \right)^2 \right] \tag{5.18}
 \end{aligned}$$

Finally, the value of $E \left[f' \left(y_i^{(l+1)} \right)^2 \right]$ depends on the activation function we used. In the case of a ReLU function its derivative is given by $f'(x > 0) = 1$ and $f'(x \leq 0) = 0$. Using the fact that $y_i^{(l+1)}$ is symmetric zero mean as we saw in section 5.2.1.

$$E \left[f' \left(y_i^{(l+1)} \right)^2 \right] = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2} \tag{5.19}$$

Gathering results from (5.17), (5.18) and (5.19) we obtaining the final expression

$$Var \left[\Delta x_j^{(l)} \right] = \frac{1}{2} n^{(l)} Var \left[W_{ij}^{(l+1)} \right] Var \left[\Delta x_i^{(l+1)} \right] \tag{5.20}$$

which for L layers leads to

$$\text{Var} \left[\Delta x_j^{(1)} \right] = \text{Var} \left[\Delta x_i^{(L-1)} \right] \prod_{l=1}^{L-2} \frac{1}{2} n^{(l+1)} \text{Var} \left[W_{ij}^{(l)} \right] \quad (5.21)$$

To avoid exponential variance growth/shrink we set

$$\text{Var} \left[W_{ij}^{(l)} \right] = 2/n^{(l+1)} \quad (5.22)$$

The expressions found for backward and forward optimal initialisation are different (5.10),(5.22). However, choosing any of them leads to the forward(backward) variance being 1 while the backward(forward) variance is still a constant [243].

5.2.2 Population Firing rate in Spiking Neural Networks

Computing the firing rate of a spiking neural network is far from trivial due to the one-sided threshold-and-reset mechanism of spiking neurons. Several approaches have been developed to obtain this under different assumptions.

These include the diffusion approximation [24], which assumes that the weights are small compared to the distance between the reset and threshold and that weights are restricted to a total of K different values which must include both positive and negative values. We can allow for larger weights as long as they are distributed following a Laplace distribution. This is called shot-noise and generalises the diffusion approximation [248]. Both of these methods are restricted to LIF neurons. For other neuron models the threshold integration method can be used [249, 250].

In addition to the particular assumption to each method, all of them assume a homogeneous (identical neurons) population of neurons and the input spikes being generated by a finite number of homogeneous Poisson processes.

Diffusion Approximation

We start with a population of LIF neurons which receive stochastic spikes which are generated following a Poisson process.

$$\frac{dv_j}{dt} = \frac{1}{\tau}(-v_j + RI_{ext}) + \sum_i \sum_{t_i^{(f)}} w_{ij} \delta(t - t_i^{(f)}), \quad v_j < V_{th} \quad (5.23)$$

We will drop the subscript j and use v to refer to the membrane potential of an arbitrary neuron in the population. Note that v is a random variable.

We are first interested in finding the probability density of the membrane potential of the population of neurons. Since the probability density may vary over time, we define $P(v, t)$ as the probability that a neuron in the unconnected population has membrane potential v at time t . We also define the *flux* $J(v, t)$ as the net fraction of neurons in this population per unit time whose membrane potential crosses v from below.

Neurons in this population receive an external current I_{ext} and stochastic spikes at rate $r_{in}(t)$ and weighted with some weights drawn from a distribution $w_{ij} \sim p_W(w)$.

If we consider a small voltage interval $[v, v + \delta v]$, the proportion of neurons in this interval at time t is $\int_v^{v+\delta v} P(v', t) dv'$. Consider now that this proportion changes over time. This changes must be equal to the net flux in the interval. In other words, the proportion of neurons that enter the interval per unit time is equal to the net flux of neurons at the extremes of the interval (those that enter from below minus the ones that leave from above). This is expressed as

$$\frac{\partial}{\partial t} \int_v^{v+\delta v} P(v', t) dv' = J(v, t) - J(v + \delta v, t) \quad (5.24)$$

Now taking the limit $\delta v \rightarrow 0$ and rearranging we obtain the continuity equation

$$\frac{\partial P(v, t)}{\partial t} + \frac{\partial J(v, t)}{\partial v} = 0 \quad (5.25)$$

We can also add the resetting and spiking mechanism to the continuity equation by considering that trajectories that reach the threshold V_{th} appear at V_r instantly and this happens at a rate $r_{out}(t)$. Note that $r_{out}(t) = J(V_{th}, t)$.

$$\frac{\partial P(v, t)}{\partial t} + \frac{\partial J(v, t)}{\partial v} = r_{out}(t) (\delta(v - V_r) - \delta(v - V_{th})) \quad (5.26)$$

We can obtain an expression for the flux as the sum of two contributions, the first one due to the incoming stochastic spiking and the second one due to the deterministic external current and neural dynamics

$$J(v, t) = J_{jump}(v, t) + J_{drift}(v, t) \quad (5.27)$$

Let's start by computing $J_{jump}(v, t)$. We are interested in the total contribution to the flux created by neurons starting at $u < v$ and jumping above v after receiving a spike. There are a total of $r_{in}(t)$ spikes per unit time each of them contributing $w \sim p_W(w)$, thus only weights $w \geq v - u$ will make neurons at u cross v . This is given by $r_{in}(t) \int_{v-u}^{\infty} p_W(w) dw$. We repeat this argument for all $u < v$, leading to

$$J_{jump}(v, t) = r_{in}(t) \int_{-\infty}^v P(u, t) \int_{v-u}^{\infty} p_W(w) dw du \quad (5.28)$$

where we have also weighted the contribution to the flux according to the original density $P(u, t)$.

The flux $J_{drift}(v, t)$ can be computed by simply considering the *velocity* $\frac{dv}{dt}$ times the

proportion of neurons at this potential

$$\begin{aligned} J_{drift}(v, t) &= \frac{dv}{dt} P(v, t) \\ &= \left[-\frac{v}{\tau} + RI_{ext} \right] P(v, t) \end{aligned} \quad (5.29)$$

where the second equality is obtained from (5.23) when not considering the random input spikes.

We now consider the particular case in which we have K synaptic types giving a probability distribution

$$p_W(w) = \sum_k \Delta_k \delta(w - w_k) \quad (5.30)$$

such that $\Delta_k > 0$ is the proportion of neurons of a given type k and $\sum_k \Delta_k = 1$. Each synaptic type contributes a total number of spikes per second given by $r_k(t) = r_{in,k}(t)\Delta_k$. Substituting in (5.28) results in

$$\begin{aligned} J_{jump}(v, t) &= \int_{-\infty}^v P(u, t) \int_{v-u}^{\infty} \sum_k r_k(t) \delta(w - w_k) dw du \\ &= \sum_k r_k(t) \int_{-\infty}^v P(u, t) \int_{v-u}^{\infty} \delta(w - w_k) dw du \\ &= \sum_k r_k(t) \int_{v-w_k}^v P(u, t) du \end{aligned} \quad (5.31)$$

Where the last equality comes from the fact that $\int_{v-u}^{\infty} \delta(w - w_k) dw$ is 1 if $v - u < w_k$ and 0 otherwise. Thus, only when $u > v - w_k$ the outer integral will be non-zero.

Substituting (5.31) and (5.29) into the continuity equation with reset (5.26) we obtain

$$\begin{aligned}
\frac{\partial P(v, t)}{\partial t} = & -\frac{1}{\tau} \frac{\partial}{\partial v} \left(\left[-v + RI_{ext} \right] P(v, t) \right) \\
& + \sum_k r_k(t) (P(v - w_k, t) - P(v, t)) \\
& + r_{out}(t) (\delta(v - V_r) - \delta(v - V_{th}))
\end{aligned} \tag{5.32}$$

Where we can identify the first term with the drift, the second with the stochastic spikes and the last one with the reset. By expanding $P(v - w_k, t)$ into a Taylor series up to a second order around v we obtain

$$\begin{aligned}
\tau \frac{\partial P(v, t)}{\partial t} = & -\frac{\partial}{\partial v} \left(\left[-v + RI_{ext} + \tau \sum_k r_k(t) w_k \right] P(v, t) \right) \\
& + \frac{1}{2} \left[\tau \sum_k r_k(t) w_k^2 \right] \frac{\partial^2}{\partial v^2} P(v, t) \\
& + r_{out}(t) (\delta(v - V_r) - \delta(v - V_{th}))
\end{aligned} \tag{5.33}$$

This is the Fokker-Plank equation of the LIF model. We define the total drive $\mu(t)$ and diffusive noise $\sigma(t)$ to be

$$\mu(t) = RI_{ext} + \tau \sum_k r_k(t) w_k \tag{5.34}$$

$$\sigma^2(t) = \tau \sum_k r_k(t) w_k^2 \tag{5.35}$$

leading to a more compact form

$$\begin{aligned}
\tau \frac{\partial P(v, t)}{\partial t} = & -\frac{\partial}{\partial v} ([-v + \mu(t)] P(v, t)) + \frac{\sigma^2(t)}{2} \frac{\partial^2}{\partial v^2} P(v, t) \\
& + r_{out}(t) (\delta(v - V_r) - \delta(v - V_{th}))
\end{aligned} \tag{5.36}$$

It can be shown that under the diffusion approximation and given constant input rates, the original LIF equation with noisy input spikes (5.23) is equivalent to an Ornstein-Uhlenbeck process given by

$$\tau \frac{dv(t)}{dt} = -v(t) + \mu + \xi(t) \quad v(t) < V_{th} \quad (5.37)$$

where $\xi(t) \sim \mathcal{N}(0, \sigma^2 \tau)$ is uncorrelated white Gaussian noise. This leads to the steady state membrane potential mean and variance

$$\mu_v = RI_{ext} + \tau \sum_k r_k w_k \quad (5.38)$$

$$\sigma_v^2 = \frac{\tau}{2} \sum_k r_k w_k^2 \quad (5.39)$$

The mean μ_v approaches this value exponentially fast (with a rate given by the membrane time constant $\exp(-t/\tau)$) and the variance σ_v^2 approaches this value somewhat faster as shown in Appendix C.

Without a threshold, the membrane distribution should be Gaussian at all times with time varying mean and variance until it reaches the steady-state distribution with parameters given by (5.38) and (5.39). However, adding the threshold and reset changes this distribution as it is equivalent to adding a boundary condition onto the Fokker-Planck equation (5.36).

The diffusion equation (5.36) does not have closed form solution in general, however, assuming that the input rates are constant $r_k(t) = r_k$, we can obtain the stationary (or steady-state) distribution. That is, the time independent distribution of membrane potential $P(v)$ which is eventually reached when $\frac{\partial P(v,t)}{\partial t} = 0$.

Using the fact that $P(v,t)$ is a probability distribution and as such it must add up to 1 and that the density above the threshold should be 0, it can be shown that the

stationary distribution is given by

$$P(v) = \begin{cases} r_{out} \frac{2\tau}{\sigma^2} \exp\left(-\frac{(v-\mu)^2}{\sigma^2}\right) \int_{V_r}^{V_{th}} \exp\left(\frac{(v-\mu)^2}{\sigma^2}\right) & v \leq V_r \\ r_{out} \frac{2\tau}{\sigma^2} \exp\left(-\frac{(v-\mu)^2}{\sigma^2}\right) \int_v^{V_{th}} \exp\left(\frac{(v-\mu)^2}{\sigma^2}\right) & V_r < v \leq V_{th} \end{cases} \quad (5.40)$$

where the output firing rate r_{out} is obtained by solving Siegert's integral

$$\frac{1}{r_{out}} = \tau \sqrt{\pi} \int_{\frac{V_r - \mu}{\sigma}}^{\frac{V_{th} - \mu}{\sigma}} \exp(x^2) (1 + \operatorname{erf}(x)) dx \quad (5.41)$$

where $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-x'^2) dx'$.

In brief, the diffusion approximation allows us to compute the expected output firing rate of a LIF neuron which receives independent stochastic input spikes through several synaptic types provided the rates are constant and the synaptic weights are small.

As a final note, these results can be extended for different weight distributions beyond simply having K synaptic types. Moreover, it is also possible to extend the diffusion approximation for other synaptic kernels beyond the delta synapses used here. In the end, the diffusion approximation only requires the mean and variance of the membrane potential. Appendix C shows how to derive them for any weight distribution and [240] shows how to obtain them for an arbitrary synaptic types.

Shot-noise

The main disadvantage of the diffusion approximation introduced before is that it requires the weights to be small. In cases where this condition is not met, the theoretical output firing rate obtained using (5.41) will not properly reflect the

empirical firing rate. Similarly the stationary membrane distribution in (5.40) will also be inaccurate.

The small weights assumption can be dropped if we consider that synaptic weights follow an exponential distribution. Namely, considering both positive and negative weights distributions

$$P_{W_e}(w) = \frac{1}{w_e} \exp\left(-\frac{w}{w_e}\right), \quad w > 0 \quad (5.42)$$

$$P_{W_i}(w) = \frac{1}{w_i} \exp\left(-\frac{w}{w_i}\right), \quad w < 0 \quad (5.43)$$

with $w_e > 0$ and $w_i < 0$. Note that if $w_e = -w_i$ we can describe both positive and negative weights with a Laplace distribution.

The total flux can be written as a combination of the drift contribution and the jump contribution of each type excitatory and inhibitory separately.

$$J(v, t) = -\frac{v}{\tau}P(v, t) + J_e(v, t) + J_i(v, t) \quad (5.44)$$

where we assumed the external current is zero.

Substituting the synaptic distributions (5.42) and (5.43) into the flux expression (5.28) we note that J_e and J_i are solutions to the following differential equations respectively

$$\begin{aligned} \frac{\partial J_e(v, t)}{\partial v} &= -\frac{J_e(v, t)}{w_e} + r_e P(v, t) - r_{out}(t) \delta(v - V_{th}) \\ \frac{\partial J_i(v, t)}{\partial v} &= -\frac{J_i(v, t)}{w_i} + r_i P(v, t) \end{aligned} \quad (5.45)$$

where r_e and r_i are the steady-state excitatory and inhibitory firing rates respectively. This is a direct consequence of choosing the weights to be exponentially

distributed. Note that we have added $r_{out}(t)$ in the excitatory term to account for the threshold.

After applying the bilateral Laplace transform on equations (5.26) and (5.45) it is possible to obtain the following expression for the stationary output firing rate

$$\frac{1}{r_{out}} = \tau \int_0^{1/w_e} \frac{Z_0^{-1}(x)}{x} \left(\frac{e^{xV_{th}}}{1-xw_e} - e^{xV_r} \right) dx \quad (5.46)$$

with $Z_0^{-1}(x) = (1-xw_e)^{\tau r_e} (1-xw_i)^{\tau r_i}$.

Moreover, we can also obtain the Laplace transform of the probability distributions. In principle it is possible to obtain compute the membrane potential probability distribution by applying the inverse Laplace transform. However, in practise it is difficult to do this and we resort to other methods to obtain it. Namely, threshold integration which is discussed in the next section.

5.2.3 Threshold integration

Threshold integration is a different method that allows to numerically find the membrane distribution and firing rate of spiking populations under the diffusion or shot noise case. This section follows the reasoning and derivations in [249].

Regardless of which neuron model we use, we always have the continuity equation and a Fokker-Planck equation which relates the time derivative of the distribution with derivatives up to a second order in the voltage

$$\frac{\partial P(v, t)}{\partial t} + \frac{\partial J(v, t)}{\partial v} = 0 \quad (5.47)$$

$$\frac{\partial P(v, t)}{\partial t} = \mathcal{L}(P(v, t)) \quad (5.48)$$

We note that if we combine and rearrange these equations we get

$$\mathcal{L}(P(v, t)) = -\frac{\partial J(v, t)}{\partial v} \quad (5.49)$$

since the operator $\mathcal{L}$ consists only on derivatives up to a second order in the voltage we can define a new operator $\mathcal{J}$ which contains only derivatives up to a first order in the voltage that relates the distribution and the flux

$$\mathcal{J}(P(v, t)) = J(v, t) \quad (5.50)$$

We have already seen some examples of the $\mathcal{J}$ operator in the drift and jump fluxes (5.28), (5.29), the simplified jump flux we used for the diffusion approximation (5.31) and, the shot noise flux (5.44) (5.45).

This allows us to obtain the stationary distribution by simply setting $\frac{\partial P(v, t)}{\partial t} = 0$ in the continuity equation and explicitly adding the boundary conditions imposed by the threshold and reset. This leads to two simultaneous first-order linear differential equations, namely the stationary continuity equation and the relationship between the membrane potential and the flux

$$\frac{\partial J(v)}{\partial v} = r_{out} (\delta(v - V_r) - \delta(v - V_{th})) \quad (5.51)$$

$$\mathcal{J}(P(v)) = J(v) \quad (5.52)$$

Note that the solutions are proportional to the steady-state firing rate giving

$$J(v) = r_{out} j(v)$$

$$P(v) = r_{out} p(v) \quad (5.53)$$

Once $\mathcal{J}$ has been established, it is possible to numerically integrate backwards from V_{th} to some lower bound voltage V_{lb} using initial conditions $j(V_{th})=1$ and $p(V_{th})=0$ (note that $J(V_{th})=r_{out}=r_{out}j(V_{th})$). Then, r_{out} can be recovered as the reciprocal of to integral $\int_{V_{lb}}^{V_{th}} p(v)dv$ since $P(v)$ must integrate to 1.

For instance, for the diffusion approximation of the LIF model, we can rearrange the Fokker-Plank equation with boundary conditions (5.36) to obtain

$$\frac{\partial P(v,t)}{\partial t} + \frac{\partial}{\partial v} \left(\frac{\mu(t)-v}{\tau} P(v,t) - \frac{\sigma^2(t)}{2\tau} \frac{\partial}{\partial v} P(v,t) \right) = r_{out}(t) (\delta(v - V_r) - \delta(v - V_{th}))$$

Then, identifying terms in (5.51) we get

$$J(v,t) = \frac{\mu(t)-v}{\tau} P(v,t) - \frac{\sigma^2(t)}{2\tau} \frac{\partial}{\partial v} P(v,t) \quad (5.54)$$

and finally in the steady-state we get the operator

$$J(v) = \mathcal{J}(P(v)) = \left(\frac{\mu-v}{\tau} - \frac{\sigma^2}{2\tau} \frac{\partial}{\partial v} \right) P(v) \quad (5.55)$$

and thus we can rearrange to obtain the following simultaneous equations

$$\frac{\partial J(v)}{\partial v} = r_{out} (\delta(v - V_r) - \delta(v - V_{th})) \quad (5.56)$$

$$\frac{\partial P(v)}{\partial v} = \frac{2\tau}{\sigma^2} \left(\frac{v-\mu}{\tau} P(v) + J(v) \right) \quad (5.57)$$

Note how using (5.53) the unknown output firing rate can be factored out resulting in two simultaneous linear differential equations that can be easily solved

$$\frac{\partial j(v)}{\partial v} = \delta(v - V_r) - \delta(v - V_{th}) \quad (5.58)$$

$$\frac{\partial p(v)}{\partial v} = \frac{2\tau}{\sigma^2} \left(\frac{v - \mu}{\tau} p(v) + j(v) \right) \quad (5.59)$$

Threshold integration can also be applied for the shot-noise case as shown in Appendix [D](#).

5.3 Firing Rate Collapse

In principle, we should be able to obtain the expected firing rate with any of the three methods mentioned in Section [5.2](#). However, these methods assume a continuous time neuron model when in fact neurons are simulated in discrete time. Thus, as we increase to simulation timestep we find that these predictions worsen.

This is hardly a problem in most of the computational neuroscience literature, as often it is perfectly feasible to simply decrease the simulation timestep. Nevertheless, as we start to deal with SNNs for deep learning the increasing memory and computational cost constrains the timestep to be about one order of magnitude larger (at around 1ms) than in the computational neuroscience literature (usually below 0.1ms). This change in temporal resolution is important enough that the effect of a larger timestep can no longer be ignored.

The following sections start by illustrating the problem and several hypotheses as to why it happens. Then, we propose three different types of solutions for it under different assumptions.

5.3.1 Firing Rate Collapse problem identification

The continuous time expression for the subthreshold membrane potential under a constant bias current and random input spikes after integrating the input for Δt seconds is given by

$$v(t) = \underbrace{\alpha v(t - \Delta t) + (1 - \alpha)RI}_{\text{Deterministic}} + \underbrace{\sum_i \sum_{t_i^{(f)}} w_i \exp\left(-\frac{t - t_i^{(f)}}{\tau_m}\right) H(t - t_i^{(f)})}_{\text{Stochastic}}, \quad v(t) < V_{th} \quad (5.60)$$

with $\alpha = \exp(-\Delta t / \tau_m)$ and $t_i^{(f)} \in [t, t + \Delta t]$.

Note that the spike integration in (5.60) requires evaluating an exponential at continuous times $t - t_i^{(f)}$, $\forall i$. This is problematic if we want to use a discrete uniform time step Δt as all spikes are synchronised to a grid with Δt resolution. Thus, we cannot generate more than one spike during a time interval either on the input or the output.

For this reason, most SNN simulators assume that all input spikes take place at a fixed point during the interval, often either at the beginning or at the end. The first thing we note is that if we assume that spikes are at the end, the exponential in (5.60) tends to 0 for large Δt . This is indeed the case for the default settings of some spiking neuron libraries like `brian2` [251]. Figure 5.1 shows that the firing rate tends to zero as we increase Δt as the effect of the input spikes becomes negligible.

Alternatively, we would expect that choosing the beginning of the interval would lead to a simulation firing rate closer to the theoretical rate, if not slightly higher, as the input spike strength has not decayed at all. Yet, this is not what we observe in Figure 5.2. Instead, the firing rate still decays with the simulation timestep in

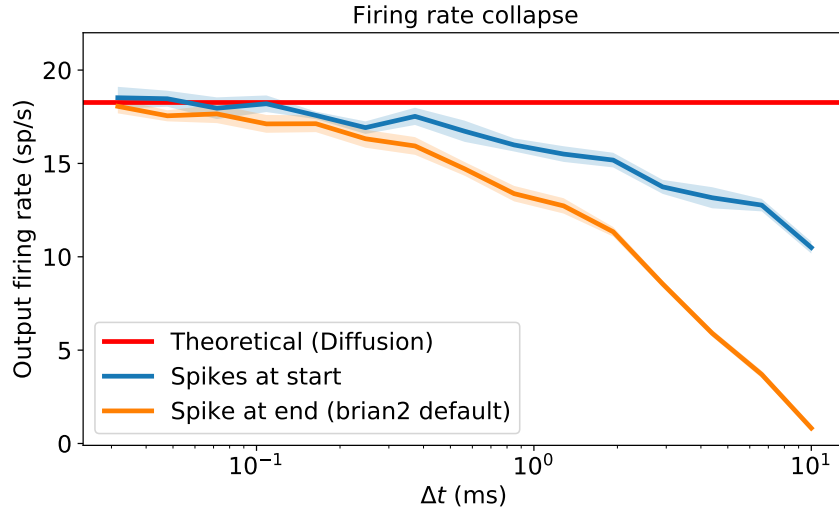


Figure 5.1: Firing rate collapse as the simulation timestep increases. Average of 10 simulations with standard error in the mean. Population of 1000 LIF neurons receiving inputs from 1000 Poisson inputs at 50Hz each connected with a probability of 0.5, the inhibitory and excitatory weights were $w_e = -w_i = 0.01$. Other parameters were set to $\tau = 10\text{ms}$, $I_{ext} = 0.8$, $R = 1$, $V_r = 0$ and $V_{th} = 1$.

both the diffusion and the shot noise case. Importantly, the firing rate collapse is already substantial at around 1ms, thus challenging any predictions that we would make for deep SNN training (particularly so on the final layers).

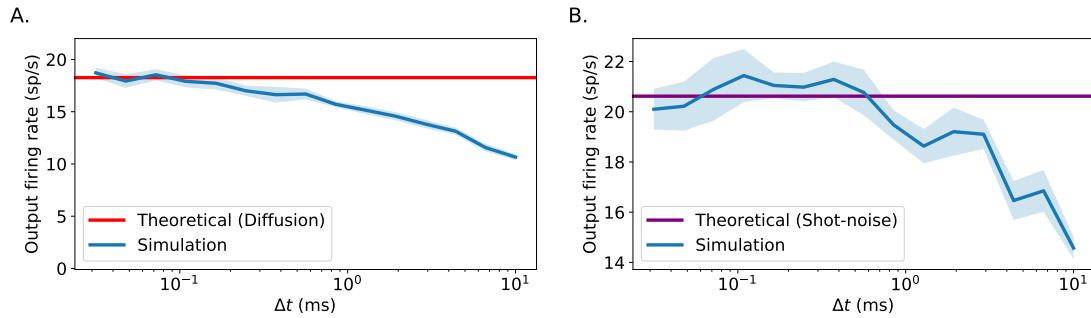


Figure 5.2: Firing rate collapse as the simulation timestep increases. Average of 10 simulations with standard error in the mean. **A.** Diffusion case. A population of 1000 LIF neurons receiving inputs from 1000 Poisson inputs at 50Hz connected with a probability of 0.5, the inhibitory and excitatory weights were $w_e = -w_i = 0.01$. Other parameters were set to $\tau = 10\text{ms}$, $I_{ext} = 0.8$, $R = 1$, $V_r = 0$ and $V_{th} = 1$. **B.** Shot-noise case with 3000 output neurons and 1600 Poisson sources at 0.14Hz connected with probability 1. Other parameters were set to $\tau = 10\text{ms}$, $I_{ext} = 0.$, $R = 1$, $V_r = 0$, $V_{th} = 1$ and $a_e = -a_i = 0.4$.

5.3.2 Hypothesis

We can propose three different hypothesis to explain why the firing rate decays as we increase the timestep length:

Hypothesis 5.1. *We cannot generate high enough input rates at large Δt because each Poisson generator can generate at most a single spike per step.*

Hypothesis 5.2. *We cannot generate high enough output rates at large Δt because each output neuron can only spike once per step.*

Hypothesis 5.3. *We are adding all inhibitory and excitatory spikes together at once but some permutation of the incoming spikes could lead to crossing the threshold before integrating the rest of the spikes.*

Hypothesis 5.1 is a problem that only takes place if the rates are too high relative to the step length. For instance, looking at 5.2A at $\Delta t = 1ms$ and given that input units fire at 50Hz, they only fire more than once per timestep 0.12% of the time. Thus, we should correct the input rate to 49.94Hz. This however leads to a theoretical output rate of 18.25Hz instead of 18.26Hz when we empirically get about 16Hz in the simulation. Thus, this hypothesis does not explain why we see such a drastic firing rate collapse.

Hypothesis 5.2 is similarly invalidated if we assume that the output neuron spike generation is also Poisson and note that the theoretical output rate of 18.26Hz is lower than that of the input leading to spike more than once per interval just 0.016% of the time.

For the final hypothesis 5.3, the key idea is that simulating all input spikes as arriving at the same time is equivalent to constraining the order of spike arrival by only considering orderings in which all excitatory inputs are perfectly aligned with an inhibitory input and thus only the difference of excitatory and inhibitory inputs

is integrated as illustrated in Figure 5.3A. Yet, other orderings which lead to spiking while ending in the same final value are still possible as shown in Figure 5.3B.

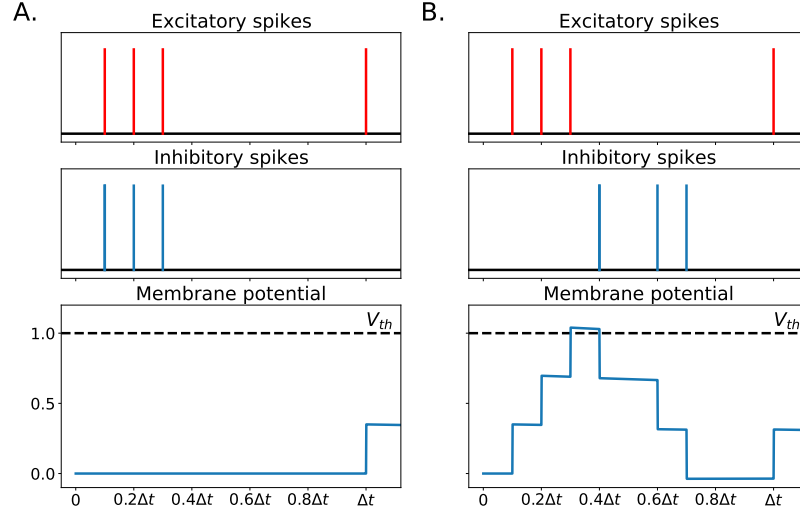


Figure 5.3: Two different input spike orderings that lead to different results. In panel A we do not spike while in panel B we do. Yet, in both cases the net input is the same $w(N - M) = w(4 - 3) = w$. Integrating all spikes at once always leads to the first case.

5.3.3 General assumptions and proposed solutions

For simplicity, since all input spikes come from independent Poisson sources we can merge them to get a single Poisson process with rate equal to the sum of the rates of each process. Then, the *actual* value of membrane potential during a time step is given by

$$v(t + s\Delta t) = \alpha(s)v(t) + (1 - \alpha(s))I + \sum_{i=1}^n w_i \exp\left(-\frac{t + s\Delta t - t_i}{\tau}\right) H(t - t_i), \quad s \in [0, 1] \quad (5.61)$$

where $\alpha(s) = \exp(-s\Delta t/\tau)$, n is the total number of spikes received during the interval and t_i is a sequence of independent uniform random variables in $[t, t + \Delta t]$.

In order to simplify our analysis we will adopt the following assumptions:

Assumption 5.1. *The spikes integrated during a timestep are not affected by the decay within that timestep.*

Assumption 5.2. *The deterministic part of the membrane potential during a timestep remains constant during that timestep and takes the value $\alpha v(t) + (1 - \alpha)I$.*

With these assumptions equation (5.61) becomes

$$v(t + s\Delta t) = \alpha v(t) + (1 - \alpha)I + \sum_{1 \leq i \leq \lfloor sn \rfloor} w_i, \quad s \in [0, 1] \quad (5.62)$$

Note that as a consequence of Assumption 5.1 the decaying exponentials multiplying the weights have been removed altogether and we are left with a piece-wise constant function where the input spikes are all equally spaced within the timestep. This may seem counter intuitive as it looks like we have lost some important timing information. However, as per Hypothesis 5.3, the key reason why we loose output spikes is due to ordering of the spikes and Equation (5.62) still allows for all possible permutations.

One could be tempted to use the average exponential value $\bar{\alpha} = \mathbb{E}(\exp(-t_i/\tau)) = \frac{\tau}{\Delta t} (1 - \exp(-\Delta t/\tau))$ leading to an effective weight of $w_i \bar{\alpha}$. Note however that $\bar{\alpha} \rightarrow 0$ as we increase Δt and thus we could end up in a similar situation as in Figure 5.1. Besides, often, threshold crossing will take place for permutations with several excitatory spikes close together, thus the effective weight will be closer to w_i , leading again to Assumption 5.1. Empirical results also showed that choosing $w_i \bar{\alpha}$ resulted in worse performance.

Similarly, we could have used the average decay value $\bar{\alpha}$ for the deterministic part. We choose to use α because the empirical results are slightly better when using this value as we will show in the following sections.

In the following three sections we analyse this problem under three different particular assumptions:

- **Two weights of the same magnitude and opposite sign:** We model the incoming spikes as a random walk which allows us to analytically compute the probability of firing.
- **Random weights and high number of spikes per interval:** We model the incoming spikes as a Wiener process which also allows us to analytically compute the probability of firing.
- **General case:** In the general case we have random weights from an arbitrary distribution and we do not constrain the spiking rate. We cannot derive an analytical solution for this case but we can devise an algorithm to simulate the correct behaviour by randomly choosing a permutation of the input spikes.

5.3.4 Input spikes as a random walk

Approximating spike integration as a random walk

We now assume that we only have two possible weight values corresponding to excitatory and inhibitory with the same magnitude, $w_{exc} = -w_{inh} = w$. Then, we can write the membrane potential within that timestep as

$$v(t + s\Delta t) = \alpha v(t) + (1 - \alpha)I + w \sum_{1 \leq i \leq \lfloor sn \rfloor} U_i, \quad s \in [0, 1] \quad (5.63)$$

where U_i is a sequence of independent random variables, each taking values 1 and -1 with probabilities $p \in [0, 1]$ and $1 - p$ respectively. Within the interval, the input is a random walk with steps of size w

$$wX_{\lfloor sn \rfloor} = w \sum_{i=1}^{\lfloor sn \rfloor} U_i \quad (5.64)$$

By the end of the interval the net input is simply the net difference of excitatory and inhibitory spikes. For instance, if the neuron receives N excitatory and M inhibitory spikes we get

$$v(t + \Delta t) = \alpha v(t) + (1 - \alpha)I + w(N - M) \quad (5.65)$$

We are interested in computing the probability that the membrane potential crosses the threshold at some point within the timestep given that at the end of the timestep interval a total of $n = N + M$ spikes were received (N excitatory and M inhibitory) and the net integrated input was $k = N - M$.

We will show that this is equivalent to asking what is the probability that the running maximum of a random walk is above a certain value y given that the end value after n steps is k .

Running maximum of a random walk

With a slight abuse of notation we will refer to the random walk $X_{[sn]}$ as X_n

$$X_n = \sum_{i=1}^n U_i, \quad n \in \mathbb{N} \quad (5.66)$$

Let also $Y_n = \max\{X_1, X_2, \dots, X_n\}$, the running maximum during the first n steps. Finding the joint probability distribution $\mathbb{P}(Y_n \geq y, X_n = k)$ is a well known problem in the random walk literature which can be easily solved if we further assume that excitatory and inhibitory spikes arrive with the same probability $p = 0.5$. In this case, the reflection principle can be used to obtain the following [252, 253]

$$\mathbb{P}(Y_n \geq y, X_n = k) = \mathbb{P}(X_n = 2y - k), \quad k \leq y \leq n \quad (5.67)$$

The case where the running maximum is less than the final value ($y < k$) is easily obtained by noticing that the event $Y_n \geq y \subset X_n = k$. Leading to

$$\mathbb{P}(Y_n \geq y, X_n = k) = \mathbb{P}(X_n = k), \quad y < k \leq n \quad (5.68)$$

These probabilities can be easily computed with

$$\mathbb{P}(X_n = k) = \begin{cases} \binom{n}{\frac{n+k}{2}} 2^{-n}, & k \in \{n, -n+2, \dots, n-2, n\} \\ 0, & \text{otherwise} \end{cases} \quad (5.69)$$

Correcting the probability of firing with a random walk

Assumption 5.3. (*Symmetric random walk*) The probability of receiving a excitatory spike is the same as that of receiving an inhibitory spike.

With the previous results we can formulate the following theorem:

Theorem 5.1. *Provided Assumptions 5.1, 5.2 and 5.3 are satisfied. The probability that a LIF neuron with time constant τ fires at least once in a timestep of length Δt given that it receives N excitatory and M inhibitory spikes each contributing $\pm w$ and an external current I is given by*

$$\mathbb{P} \left(\max_{s \in [t, t+\Delta t]} v(s) \geq V_{th} \mid v(t + \Delta t) \right) = \begin{cases} \frac{\mathbb{P}(X_n = 2y - (N - M))}{\mathbb{P}(X_n = N - M)}, & N - M \leq y \leq n \\ 1, & y < N - M \leq n \\ 0, & y > N \end{cases} \quad (5.70)$$

where $n = N + M$, $y = \left\lceil \frac{V_{th} - (\alpha v(t) + (1 - \alpha)I)}{w} \right\rceil$, with $\alpha = \exp(-\Delta t / \tau)$ and X_n being a symmetric random walk.

Proof.

Cases 1 and 2:

The deterministic part of the membrane potential during the interval $[t, t + \Delta t]$ is $\alpha v(t) + (1 - \alpha)I$. Then, the number of excitatory inputs that a LIF neuron at potential $\bar{v}$ needs to cross the threshold is given by

$$y = \left\lceil \frac{V_{th} - (\alpha v(t) + (1 - \alpha)I)}{w} \right\rceil \quad (5.71)$$

Thus, the probability of crossing the threshold is the same as the probability of the running maximum being greater than y

$$\mathbb{P} \left(\max_{s \in [t, t + \Delta t]} v(s) \geq V_{th} \right) = \mathbb{P} \left(Y_n = \max_i X_i \geq y \right) \quad (5.72)$$

Since we know that the random walk ends at $X_n = N - M$ we have

$$\begin{aligned} \mathbb{P} \left(\max_{s \in [t, t + \Delta t]} v(s) \geq V_{th} \mid v(t + \Delta t) \right) &= \mathbb{P} \left(Y_n \geq y \mid X_n = N - M \right) \\ &= \frac{\mathbb{P}(Y_n \geq y, X_n = N - M)}{\mathbb{P}(X_n = N - M)} \\ &= \begin{cases} \frac{\mathbb{P}(X_n = 2y - (N - M))}{\mathbb{P}(X_n = N - M)}, & N - M \leq y \leq n \\ \frac{\mathbb{P}(X_n = N - M)}{\mathbb{P}(X_n = N - M)} = 1, & y < N - M \leq n \end{cases} \end{aligned} \quad (5.73)$$

Case 3:

If the number of steps required is less than the total number of excitatory inputs $N < y$ then the neuron will never cross the threshold in this timestep. Thus, the

probability of spike is 0. □

Note that a regular simulation where we consider all spikes taking place at the beginning is equivalent to a probability of spike according to Cases 2 and 3 only. The correction we add corresponds to Case 1, where a neuron that did not spike according to the final value $v(t + \Delta t)$ still could have spiked in between.

Implementing this correction is rather simple. We can perform the integration as usual and simply take those neurons that did not spike but could have spiked, compute their probability of spiking according to (5.70) and randomly sample whether they spike. This is shown in Figure 5.4. Failure after $\Delta t = 20ms$ is due to the 50Hz Poisson generators no longer able to keep up (as in Hypothesis 5.1).

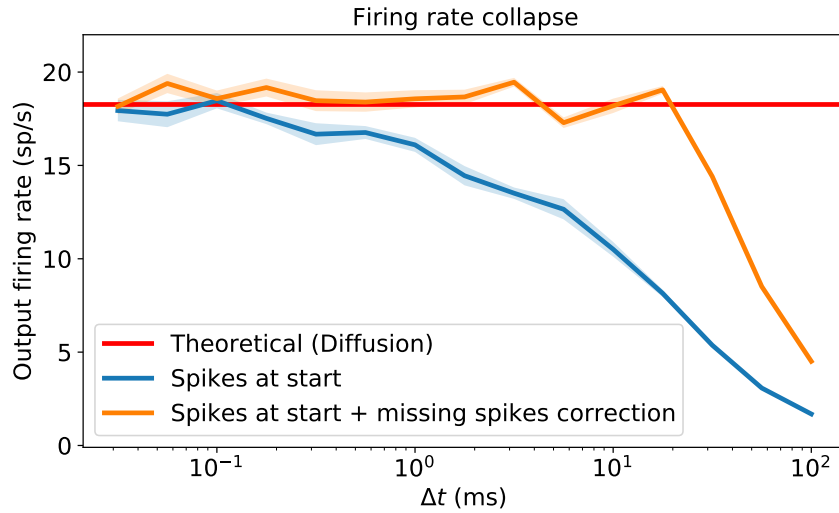


Figure 5.4: Firing rate collapse as the simulation timestep increases and correction. Simulation details are identical to Figure 5.2. The integration is performed identically but neurons that did not spike but could have spiked are made to spike randomly according to Theorem 5.1. Failure after $\Delta t = 20ms$ is due to the 50Hz Poisson generators no longer able to keep up.

If we were to use the average value of α for the computation of y we get the slightly worse result in Appendix E.

We can easily extend the result in Theorem 5.1 to time varying thresholds $V_{th}(t)$ and non-linear membrane potential equations with the following result.

Theorem 5.2. *Under the same assumptions as Theorem 5.1, a non-linear membrane equation of the form $\frac{dv}{dt} = f(v) + \sum_i w_i \delta(t - t_i)$ where the deterministic part is discretised with $g(v)$ and $w_i \in \{+w, -w\}$, and given a time-varying threshold $V_{th}(t)$ the results in Theorem 5.1 hold for $y = \left\lceil \frac{V_{th}(t) - g(v(t))}{w} \right\rceil$.*

Proof. *The proof is identical to that of Theorem 5.1 upon substitution of the threshold and discretisation expressions.* □

5.3.5 Input spikes as a Wiener process

An important limitation of the previous results is that it restricts us to only use two symmetric excitatory and inhibitory weights as well as requiring an identical excitatory and inhibitory rates. A simple workaround this can be achieved by considering that the input rates are large enough during a timestep so that we can approximate the (now not necessarily symmetric) random walk to a Wiener process. This is similar to the diffusion approximation requirements with the subtle difference that the input rate must be high enough so that number of spikes received in a timestep is large enough.

Wiener Process as a limit of a random walk

A Wiener process W_t is a random process characterised by

1. $W_0 = 0$
2. It has independent increments. $W_{t+s} - W_t, s \geq 0$ is independent from $W_u, u < t$
3. It has Gaussian increments $W_{t+s} - W_t \sim \mathcal{N}(0, s)$
4. It is continuous in t .

The probability density function of a Wiener process follows a Gaussian distribution with mean 0 and variance t

$$f_{W_t}(w) = \frac{1}{\sqrt{2\pi t}} \exp\left(-\frac{w^2}{2t}\right) \quad (5.74)$$

A standard result due to Donsker [254] states that given a sequence of independent identically distributed random variables $\xi_1, \xi_2, \dots, \xi_n$, with mean 0 and variance 1 the random walk:

$$W_n(t) = \frac{1}{\sqrt{n}} \sum_{1 \leq k \leq \lfloor tn \rfloor} \xi_k, \quad t \in [0, 1] \quad (5.75)$$

converges in distribution to a Wiener process as $n \rightarrow \infty$. This result explains why Wiener processes are good models for many stochastic phenomena in nature. In fact, the diffusion approximation we introduced in section 5.2.2 uses this result to approximate the input spikes as Gaussian noise.

Running maximum of a Wiener process

We define the running maximum of a Wiener process as

$$M_t = \max_{0 \leq s \leq t} W_s \quad (5.76)$$

The reflection principle can also be used in a Wiener process to obtain the joint density function (see [255][256] for a derivation)

$$f_{M_t, W_t}(m, w) = \frac{2(2m - w)}{t\sqrt{2\pi t}} \exp\left(-\frac{(2m - w)^2}{2t}\right), \quad m \geq 0, w \leq m \quad (5.77)$$

With (5.74) and (5.77) we can compute the conditional density function

$$f_{M_t|W_t}(m|w) = \frac{f_{M_t, W_t}(m, w)}{f_{W_t}(w)} = \frac{2(2m - w)}{t} \exp\left(-\frac{2m(m - w)}{t}\right), \quad m \geq 0, w \leq m \quad (5.78)$$

Then, we can compute the conditional probability

$$\mathbb{P}(M_t \geq m|W_t) = \exp\left(-\frac{2m(m - W_t)}{t}\right), \quad m > \max(0, W_t) \quad (5.79)$$

Correcting the probability of firing with a Wiener process

With the previous results we can formulate the following Theorem.

Theorem 5.3. *Provided Assumptions 5.1 and 5.2 are satisfied. Then, the probability that a LIF neuron with time constant τ receiving an external current I fires at least once in a timestep of length Δt given that the weight distribution has mean μ_w and variance σ_w , and that by the end of the interval a net contribution of W due to the incoming spikes was integrated, converges in distribution to*

$$\mathbb{P}\left(\max_{s \in [0, 1]} v(t + s\Delta t) \geq V_{th} \mid v(t + \Delta t)\right) = \begin{cases} \exp\left(-\frac{2m(m - W_{t+\Delta t})}{\Delta t}\right), & m > \max(0, W_{t+\Delta t}) \\ 1, & m \leq W_{t+\Delta t} \end{cases} \quad (5.80)$$

as the number of input spikes per interval n increases, with $m = \frac{V_{th} - (\alpha v(t) + (1 - \alpha)I) - \mu_w}{\sigma_w \sqrt{n/\Delta t}}$ and $W_{t+\Delta t} = \frac{W - \mu_w}{\sigma_w \sqrt{n/\Delta t}}$.

Proof.

The membrane potential during the timestep is given by

$$v(t + s\Delta t) = \alpha v(t) + (1 - \alpha)I + \sum_{1 \leq i \leq \lfloor sn \rfloor} w_i, \quad s \in [0, 1] \quad (5.81)$$

We can rewrite the stochastic part of the membrane potential as

$$\sum_{1 \leq i \leq \lfloor sn \rfloor} w_i = \mu_w + \sigma_w \sum_{1 \leq i \leq \lfloor sn \rfloor} \xi_i \quad (5.82)$$

where now ξ_i has zero mean and unit variance. Then, according to Donsker Theorem [254],

$$W_n(s) = \frac{1}{\sqrt{n}} \sum_{1 \leq i \leq \lfloor sn \rfloor} \xi_i, \quad s \in [0, 1] \quad (5.83)$$

converges in distribution to a Wiener process W_s as the number of input spikes tends to infinity. Since our interval has a length of Δt we simply have to rescale the process $W_{t+s\Delta t} = \sqrt{\Delta t} W_s$ and define $W_t = 0$. Then, we can rewrite (5.81) as

$$v(t + s\Delta t) = \alpha v(t) + (1 - \alpha)I + \mu_w + \sigma_w \sqrt{\frac{n}{\Delta t}} W_{t+s\Delta t}, \quad s \in [0, 1] \quad (5.84)$$

This means that for the neuron to spike we need the running maximum of the Wiener process of be larger than

$$m = \frac{V_{th} - (\alpha v(t) + (1 - \alpha)I) - \mu_w}{\sigma_w \sqrt{n/\Delta t}} \quad (5.85)$$

That is

$$\mathbb{P} \left(\max_{s \in [0,1]} v(t + s\Delta t) \geq V_{th} \right) = \mathbb{P} \left(\max_{s \in [0,1]} W_{t+s\Delta t} \geq m \right) \quad (5.86)$$

Finally, since we know that at the end of the interval a net contribution of W has been integrated we have that

$$W_{t+\Delta t} = \sqrt{\Delta t} W_n(t + \Delta t) = \frac{W - \mu_w}{\sigma_w \sqrt{n/\Delta t}} \quad (5.87)$$

Which leads to

$$\begin{aligned} \mathbb{P} \left(\max_{s \in [0,1]} v(t + s\Delta t) \geq V_{th} \middle| v(t + \Delta t) \right) &= \mathbb{P} \left(\max_{s \in [0,1]} W_{t+s\Delta t} \geq m \middle| W_{t+\Delta t} \right) \\ &= \begin{cases} \exp \left(-\frac{2m(m-W_{t+\Delta t})}{\Delta t} \right), & m > \max(0, W_{t+\Delta t}) \\ 1, & m \leq W_{t+\Delta t} \end{cases} \end{aligned} \quad (5.88)$$

The second case is evident from the fact that if $m \leq W_{t+\Delta t}$ then the final value is a running maximum and it is above the threshold. $\square$

We can easily apply this correction by integrating as usual and taking the neurons that did not spike, compute their probability of spiking according to (5.80) and randomly sample whether they spike. This is shown in Figure 5.5. Failure after $\Delta t = 20ms$ is due to the 50Hz Poisson generators no longer able to keep up (as in Hypothesis 5.1). We also observe that the correction is worse for small time steps as we would expect since there are not enough input spikes per timestep.

As in the random walk case, this result can be extended to more complex membrane models and time-varying threshold.

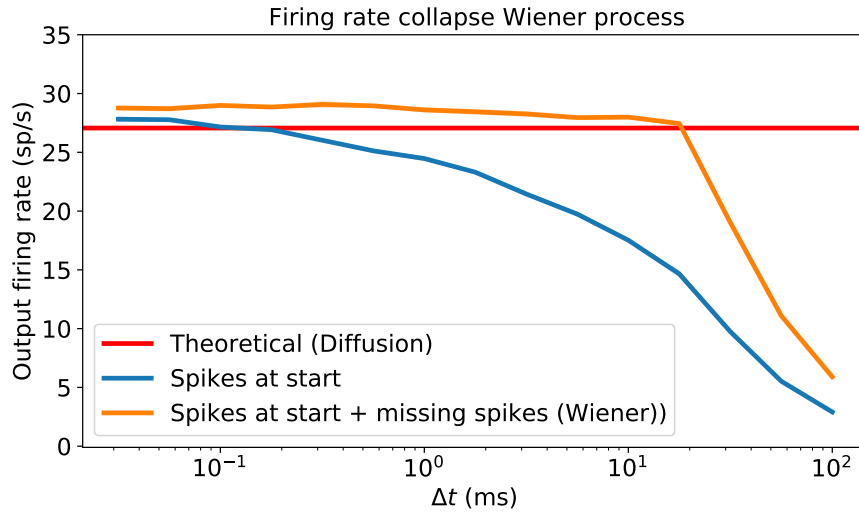


Figure 5.5: Firing rate collapse as the simulation timestep increases and correction. Average and error in the mean for 5 samples are plotted. . Simulation details are identical to Figure 5.2 but now there are 2000 inputs, the probability of connection is 1 and the weights are drawn from a Gaussian distribution with mean 0 and standard deviation 0.01. The integration is performed identically but neurons that did not spike but could have spiked are made to spike randomly according to Theorem 5.3. Failure after $\Delta t = 20ms$ is due to the 50Hz Poisson generators no longer able to keep up.

Theorem 5.4. *Under the same assumptions of Theorem 5.3, a non-linear membrane equation of the form $\frac{dv}{dt} = f(v) + \sum_i w_i \delta(t - t_i)$ where the deterministic part is discretised with $g(v)$ and given a time-varying threshold $V_{th}(t)$ the results in Theorem 5.3 hold for $m = \frac{V_{th}(t) - g(v(t)) - \mu_w}{\sigma_w \sqrt{n}}$.*

Proof. *The proof is identical to that of Theorem 5.3 upon substitution of the threshold and discretisation expressions.* □

5.3.6 General case

For the general case where we do not make any assumption about either the weights or the input rates we cannot easily derive an analytical expression. Yet, we can design an algorithm where we sample a permutation of the input spikes.

Algorithm 1 A simulation of a single time step update for the general case

```

 $v_{det} = \alpha v[t] + (1 - \alpha)I$ 
 $v[t + 1] = v_{det} + \sum_{i=1}^n w_i$ 
if  $v[t + 1] \geq V_{th}$  then
    The neuron spikes
     $v[t + 1] = V_r$ 
else
     $v_{sto} = \text{accumulate}(\text{permute}([w_1, w_2, \dots, w_n]))$ 
    if  $v_{det} + v_{sto} \geq V_{th}$  then
        The neuron spikes
         $v[t + 1] = V_r$ 
    else
        The neuron does not spike
    end if
end if

```

The function `accumulate` simply performs a cumulative sum of the input sequence.

We can parallelise this algorithm to be executed for several neurons by simply zero padding the input weights to meet the sequence of the neurons which received the most inputs in that time step. The results are shown in Figure 5.6.

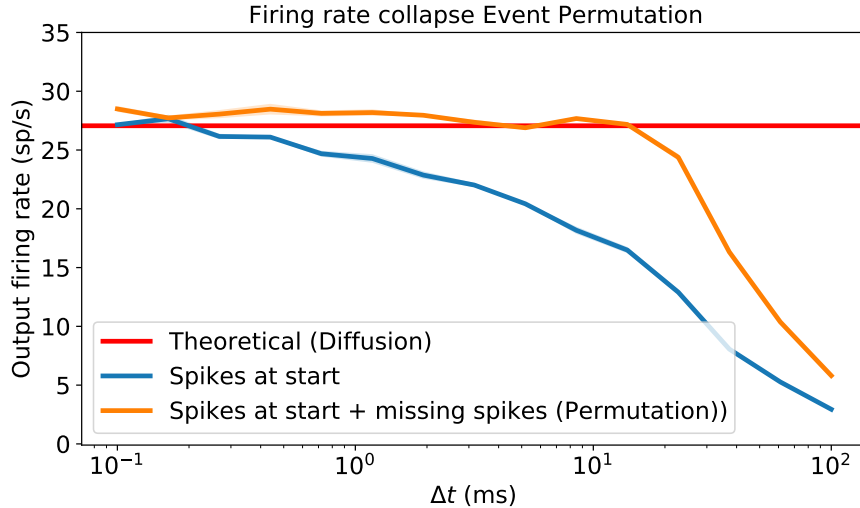


Figure 5.6: Firing rate collapse as the simulation timestep increases and correction. Average and error in the mean for 5 samples are plotted. Simulation details are identical to Figure 5.5. The integration is performed according to Algorithm 1. Failure after $\Delta t = 20ms$ is due to the 50Hz Poisson generators no longer able to keep up.

5.4 Vanishing-Exploding gradients on SNNs

After correcting the rate collapse problem, the Fokker-Plank or shot-noise approaches should be enough to ensure a proper network initialisation in the forward pass. Yet, adjusting the network firing rates is not enough to ensure that the gradients will not vanish or explode. To this end, we now attempt to use the variance propagation method for ANNs introduced in section 5.2.1 on SNNs.

Firstly, while strictly not necessary, we still include a section on applying this method to the forward pass and show that we can only get a weak bound on the firing rate at the output layer. Secondly, we attempt the same for the backward pass and obtain a closed form expression that depends on the membrane potential distribution which we can compute either by Fokker-Plank or threshold integration.

5.4.1 Variance propagation in the forward pass of an SNN

We consider a total of L spiking layers with $n^{(l)}$ neurons in each layer $l = 0, \dots, L-1$.

We use the unrolled membrane and spike expressions in discrete time

$$v_j^{(l+1)}[t+1] = \sum_{k=0}^t \alpha^{t-k} \left(\sum_i w_{ij}^{(l)} s_i^{(l)}[k] + (1-\alpha)RI \right) \quad (5.89)$$

$$s_j^{(l+1)}[t+1] = H(v_j^{(l+1)}[t+1] - V_{th}) \quad (5.90)$$

We adopt the following assumptions:

Assumption 5.4. *Membrane potentials $v_i^{(l)}$ across neurons at the same layer are independent and identically distributed.*

Assumption 5.5. *The weight distribution is symmetric and $E[w_{ij}^{(l)}] = 0$, $\forall i, j, l$*

Assumption 5.6. *Weights $w_{ij}^{(l)}$ and spikes $s_i^{(l)}[k]$ at layer l are independent.*

Assumption 5.7. *Spike times are uncorrelated at different time steps, implying $\text{Cov}[s_i^{(l)}[k], s_i^{(l)}[p]] = \text{Var}[s_i^{(l)}[k]]\delta_{pk}$.*

Assumption 5.8. *The probability of spike is the same and constant for neurons in the same layer and given by $P(s_i^{(l)}[k] = 1) = \rho^{(l)}$ and $P(s_i^{(l)}[k] = 0) = 1 - \rho^{(l)}$.*

Assumptions 5.4 through 5.7 are analogous to the assumptions used in the variance propagation in ANNs as in 5.2.1 and [243].

Assumption 5.8 implies that, given the timestep Δt is small enough, all spike trains in the network are Poisson with a rate $r^{(l)}$ obeying $\rho^{(l)} = r^{(l)}\Delta t$. Note that this assumption would mean that the network cannot carry information in the precise spike times, however, this is only required for initialisation (i.e. before any training has taken place).

With these assumption we can formulate the following result:

Theorem 5.5. *Given assumptions 5.4, 5.5, 5.6 and 5.7 are satisfied and further assuming that the membrane potential distribution is symmetric around zero, the probability that a neuron fires within a timestep is bounded according to*

$$\rho^{(l+1)} < \frac{n^{(l)} \text{Var}[w_{ij}^{(l)}]}{2V_{th}^2} \left(A(\alpha)\rho^{(l)} + B(\alpha)(\rho^{(l)})^2 \right) \quad (5.91)$$

with $A(\alpha) = \frac{1}{1-\alpha^2}$, $B(\alpha) = \frac{2\alpha}{(1-\alpha)^2(1+\alpha)}$.

Proof. *The proof can be found in Appendix F*

There are several issues with this approach. Firstly, the symmetry of the membrane potential distribution assumption is in general too strong (see 5.2.2) and will not be true for most parameter combinations. Secondly, this approach only offers an upper

bound on the firing rate, not a lower bound, meaning we can only ensure the firing rate in the final layer will be lower than a certain value. This may of course lead to no firing at all. Finally, as we add more layers, the bound becomes weaker and weaker and thus it becomes less useful.

It can be argued that other inequalities like Canteli's inequality do not require the symmetry assumption, however, it is weaker than Chebyshev's and thus the third issue remains. Even if we use other stronger inequalities we would still have the second issue, namely, we only bound from above.

The problem stems from the fact that we are trying to find a relationship between the variance of the membrane potential and the firing rate. However, we know that this is not possible without considering the boundary conditions due to the resetting as we saw in 5.2.2.

In brief, the variance propagation method applied to the forward pass on SNNs does not give a useful solution to the network initialisation problem and other methods based on the Fokker-Plank equation or shot-noise are required instead.

5.4.2 Variance propagation in the backward pass of an SNN

We assume that the loss function applied to the last layer $\varepsilon : \mathbb{R}^{n^{(L-1)}} \rightarrow \mathbb{R}$ is a function of the sum of membrane potentials in the last layer across time (i.e. $\varepsilon(v^{(L-1)}[t]) = \mathcal{L}(\sum_t v^{(L-1)}[t])$). For clarity, we will use $\frac{\partial \varepsilon}{\partial z} := \Delta z$ in this section as in [243]. The gradients of the loss function with respect the spikes at layer l are

$$\Delta s^{(l)}[t] = \sum_{k>t} \alpha^{k-t-1} \Delta v^{(l+1)}[k] W^{(l)} \quad (5.92)$$

$$\Delta v_i^{(l+1)}[t] = \Delta s_i^{(l+1)}[t] H' \left(v_i^{(l+1)}[t] \right) \quad (5.93)$$

where $\Delta s^{(l)}[t] \in \mathbb{R}^{1 \times n^{(l)}}$, $\Delta v^{(l+1)}[t] \in \mathbb{R}^{1 \times n^{(l+1)}}$, $W^{(l)} \in \mathbb{R}^{n^{(l+1)} \times n^{(l)}}$ and $H'(\cdot)$ is a function that subtracts the threshold V_{th} and then applies the surrogate derivative of the Heaviside step function. See Appendix A for a derivation of these expressions.

We adopt the following assumptions:

Assumption 5.9. *The weight distribution has expectation $E[w_{ij}^{(l)}] = 0$, $\forall i, j, l$*

Assumption 5.10. *The weights $w_{ij}^{(l)} = (W^{(l)})_{ij}$ and the gradients $\Delta v^{(l+1)}[t]$ are independent.*

Assumption 5.11. *The gradient $\Delta s_i^{(l+1)}[t]$ and the surrogate $H'(v_i^{(l+1)}[t])$ are independent.*

Assumption 5.12. *Gradients $\Delta s_i^{(l)}[t]$ that are far apart in time are independent.*

Assumption 5.13. *The surrogate function decreases fast enough so that if $H'(v_i^{(l+1)}[t]) \neq 0$ at a given t it will be close to 0 at nearby steps.*

Assumptions 5.10 and 5.11 are of course incorrect since the weights at layer l are used to compute the loss function and consequently gradients of the loss cannot be independent from the weights. Yet, these assumptions are commonly adopted in the weight initialisation literature with good empirical results (see section 5.2.1 and [243]).

Intuitively assumptions 5.12 and 5.13 mean that either neurons are not active (i.e. not close to the spiking threshold) for more than a single time step or in case they are the surrogate gradient is comparably very small at contiguous steps. These assumptions together with the fact that $\Delta v_i^{(l+1)}[k] = \Delta s_i^{(l+1)}[k] H'(v_i^{(l+1)}[k])$ mean that $Cov[\Delta v_i^{(l+1)}[k], \Delta v_i^{(l+1)}[p]] = Var[\Delta v_i^{(l+1)}[k]] \delta_{kp}$.

Theorem 5.6. *Provided assumptions 5.10 through 5.11 are satisfied, the variance of the gradient of the loss function with respect to the spikes is given by*

$$\text{Var} \left[\Delta s_j^{(l)}[t] \right] = \frac{\rho_a^{(l+1)}}{1 - \alpha^2} n^{(l+1)} \text{Var} \left[w_{ij}^{(l)} \right] \text{Var} \left[\Delta s_i^{(l+1)}[t] \right] \quad (5.94)$$

$$\text{where } \rho_a^{(l)} = E \left[\left(H' \left(v_i^{(l+1)}[t] \right) \right)^2 \right].$$

Proof. The proof can be found in Appendix F

This result is an expression of the variance of the gradients as a function of the variance of the next layer. With this, we can obtain an expression for the variance of the gradient on the first layer

$$\text{Var} \left[\Delta s_j^{(0)} \right] = \text{Var} \left[\Delta s_j^{(L-1)} \right] \prod_{l=1}^{L-1} \frac{n^{(l)} \text{Var} \left[w_{ij}^{(l-1)} \right] \rho_a^{(l)}}{1 - \alpha^2} \quad (5.95)$$

Which leads to setting

$$\text{Var} \left[w_{ij}^{(l-1)} \right] = \frac{1 - \alpha^2}{n^{(l)} \rho_a^{(l)}} \quad (5.96)$$

The challenge now is to find how to compute $\rho_a^{(l)} = E \left[\left(H' \left(v_i^{(l)}[t] \right) \right)^2 \right]$.

5.4.3 Computing $\rho_a^{(l)}$

We need to be careful when computing $E \left[\left(H' \left(v_i^{(l)}[t] \right) \right)^2 \right]$. One may be tempted to think that we can obtain the probability density function of the membrane potential $P(v)$ in the steady-state using either the diffusion approximation or the threshold integration method and then simply integrate $\int_{-\infty}^{\infty} (H'(v))^2 P(v) dv$

However, so far we have been using the membrane potential, v , in the diffusion, shot-noise and threshold integration methods as if it was identical to that defined

for the neuron update (e.g. (5.61)). These are not the same, however, since the membrane in neuron updates equation has not yet applied the threshold-and-reset mechanism. When we apply the threshold function to the membrane potential we do so *after* updating it with the incoming spikes and currents within a timestep and right *before* applying the reset. Thus, we do not apply the surrogate function to the steady state distribution but to what we will call the *before-reset distribution*. That is, the distribution we obtain after a time step Δt has taken place with initial condition $P(v)$ but before resetting the neurons.

For instance, in the diffusion case, in the absence of a spike-and-reset mechanism the membrane potential distribution evolves according to the Fokker Planck equation

$$\tau \frac{\partial P(v, t)}{\partial t} = - \frac{\partial}{\partial v} ([-v + \mu] P(v, t)) + \frac{\sigma^2}{2} \frac{\partial^2}{\partial v^2} P(v, t) \quad (5.97)$$

To obtain the *before-reset distribution* we just need to find $\hat{P}(v) = P(v, t_0 + \Delta t)$ given an initial condition $P(v, t_0) = P(v)$. A similar result can be obtained using the shot-noise equations but instead of a partial differential equation (PDE) we would had a system of differential algebraic equations (DAE) which most numerical solvers struggle with, an exception being MATLAB's PDE solver².

Once the before-reset distribution $\hat{P}(v)$ has been found we can simply compute

$$\rho_a^{(l)} = \int_{-\infty}^{\infty} (H'(v))^2 \hat{P}(v) dv \quad (5.98)$$

For the special case where our surrogate function is given by $H'(v) = H(v - (V_{th} - B_{th})) - H(v - (V_{th} + B_{th}))$ as in Chapter 4 we have

²See <https://uk.mathworks.com/help/matlab/math/partial-differential-equations.html>

$$\rho_a^{(l)} = \int_{V_{th}-B_{th}}^{V_{th}+B_{th}} \hat{P}(v) dv \quad (5.99)$$

Provided that B_{th} is small as required by Assumption 5.13, we can interpret $\rho_a^{(l)}$ as the probability of the backward spike gradient to be non-zero or if we prefer that the backward gradient ‘fires’ as we saw in Chapter 4, Figure 4.3B.

Figure 5.7 shows the membrane potential distributions after reset ($P(v)$) and before reset ($\hat{P}(v)$) for both the Wiener corrected forward pass (bottom row) and the original forward pass (top row). Note how the corrected version results in a much better approximated distribution around the threshold, necessary for computing $\mathcal{I}$.

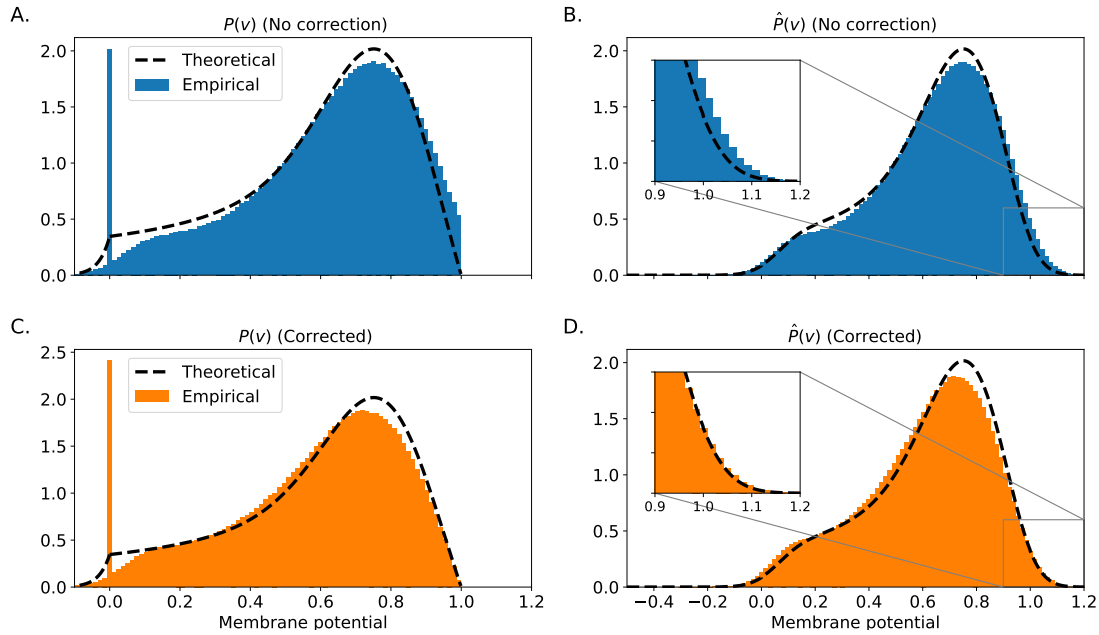


Figure 5.7: Membrane distributions theoretical and empirical. The theoretical distribution $P(v)$ was obtained by using the membrane distribution equation according to the diffusion approximation (5.40) and the before reset distribution was obtained by solving (5.97) using the `py-pde` package [257]. Note how the distribution on the corrected case fits much better around the threshold, leading to a better approximation of $\mathcal{I}$. The parameters for this simulations were $\Delta t = 1\text{ms}$, $\tau = 10\text{ms}$, $I_{ext} = 0.9$, $R = 1$, $V_r = 0$, $V_{th} = 1$, $\mu_w = 0$, $\sigma_w = 0.0096$ and $\nu_{in} = \nu_{target} = 30\text{Hz}$.

5.5 General SNN Initialisation strategy

The aim of this chapter was to find a way to initialise the weights of an SNN to ensure a certain firing rate at the output while avoiding having exploding or vanishing gradients. After the results from the previous sections we have the following:

1. We can compute the firing rate at each layer by using either the diffusion or the shot noise approximation plus using an appropriate firing rate collapse solution if necessary.
2. We have an expression for what the desired weight variance should be in order to avoid vanishing or exploding gradients.

The question now is how can we use these results to obtain a desired firing rate while ensuring the gradients will flow.

Let $\mathcal{V}$ be a operator that computes the firing rate at a layer l given the variance of the weights $\sigma_{w^{(l)}}^2$ and other parameters $\boldsymbol{\theta}$ (threshold, input rates, time constant etc). This algorithm can be Siegert's integral, shot-noise rate equation or threshold integration.

$$\nu^{(l)} = \mathcal{V}(\sigma_w^2, \boldsymbol{\theta}) \quad (5.100)$$

Let $\mathcal{W}$ be the operator that computes the optimal weight variance $\hat{\sigma}_w^2$ for the gradients to flow. This operator will include several steps: computing the stationary distribution, computing the before-reset distribution and finally computing the weight variance with (5.96). Note that one of the input parameters of $\mathcal{W}$ will be the initial variance $\sigma_{w^{(l)}}^2$.

$$\hat{\sigma}_w^2 = \mathcal{W}(\sigma_w^2, \boldsymbol{\theta}) \quad (5.101)$$

This two variances, σ_w^2 and $\hat{\sigma}_w^2$, will not necessarily be the same, but we would like for them to be equal. This is easily achieved by allowing the surrogate function to be scaled by a factor $\kappa H'(v)$. Then, equation (5.98) becomes

$$\begin{aligned}\rho_a^{(l)} &= \kappa^2 \int_{-\infty}^{\infty} (H'(v))^2 \hat{P}(v) dv \\ &= \kappa^2 \mathcal{I}\end{aligned}\tag{5.102}$$

substituting into (5.96) the expression for the backward optimal variance becomes

$$\sigma_w^2 = \frac{1 - \alpha^2}{n^{(l)} \kappa^2 \mathcal{I}}\tag{5.103}$$

and finally we can obtain an expression for κ

$$\kappa = \sqrt{\frac{1 - \alpha^2}{n^{(l)} \sigma_w^2 \mathcal{I}}}\tag{5.104}$$

We note that the integral $\mathcal{I}$ needs to be only computed once since it is fully determined by the forward pass.

Thus, the general strategy for initialising and SNN consists of the following two steps:

1. Set a desired firing rate and obtain σ_w^2 via a gradient-free root finding algorithm to solve $\nu^{(l)} - \mathcal{V}(x, \boldsymbol{\theta}) = 0$ for x .
2. Compute κ according to equation (5.104).

Since the surrogate gradient does not affect the computation of the forward firing rate it is safe to modify it without affecting the forward pass. As opposed to [240], the

scaling we apply to the surrogate function considers the statistics of the membrane potential (through $\mathcal{I}$), thus, we do not expect the gradients to explode as in their case. Moreover, since it has been found that surrogate gradient descent is remarkably robust to the surrogate function we expect that training will take place without issue [145].

We apply this method and obtain the results depicted in Figure 5.8. As we see, as long as the firing rate collapse correction is added (in this case the random walk version) the firing rate remains around the target rate quite reliably. Interestingly, at high enough rates the firing rate collapse does not go to zero but converges to some value (Figure 5.8 left). It may be interesting to study this further as predicting this value could remove the need to use the firing rate collapse correction altogether. In addition, we found that trying lower target rates results in the network biasing towards a slightly higher rate (see Appendix G) this also needs to be studied further.

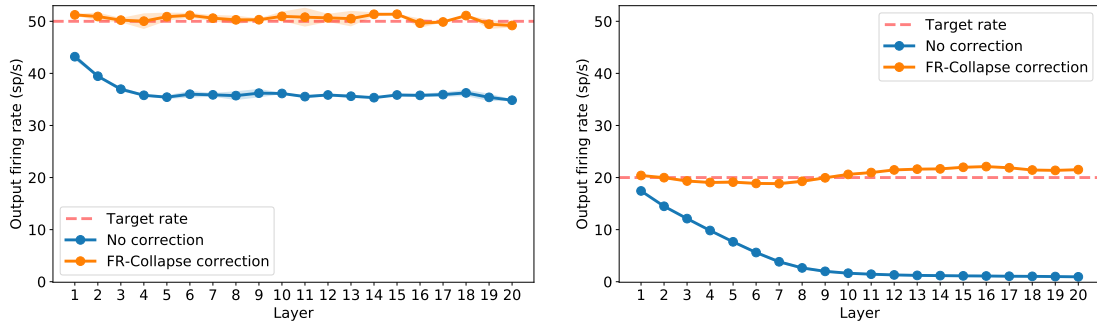


Figure 5.8: Firing rate at each layer on a feed-forward SNN with 20 layers with target rates 50Hz and 20Hz respectively. We plot the mean and standard error in the mean for 5 trials. The input spikes were Poisson trains at the target rate. Each layer consisted of 2000 LIF neurons with parameters $\tau = 10\text{ms}$, $I = 0.6$, $V_r = 0$, $V_{th} = 1$. The probability of connection between neurons in consecutive layers was 0.5. The operator $\mathcal{V}$ was the diffusion rate. The optimal weights found were 0.0387 and 0.0299 respectively. Optimisation was done using `scipy.root_scalar` function with `toms748` method. The network was simulated for 1s with timestep $\Delta t = 1\text{ms}$.

Finally, Figure 5.9 shows the gradient at each layer in a network where the forward correction was done using the Wiener method. We compare the variance of the spike gradient Δs_i with and without the κ correction. As we see, the original gradient

vanishes while the κ -corrected gradient remains nearly constant. We note however, that the variance still decays to some extent due to the fact that our assumptions are not completely correct. In fact, using a larger value of B_{th} results in a much worse variance propagation due to Assumption 5.13 not being true and consequently the covariance term between gradient being too large (see Appendix G). Yet, as we showed in Chapter 4, a small value of B_{th} (e.g. $B_{th} = 0.01$ as in the figure) is still perfectly fit for training successfully.

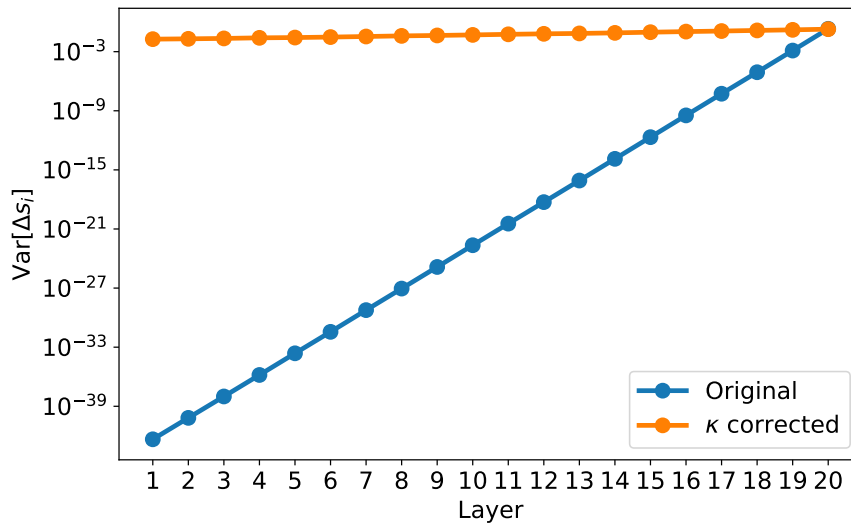


Figure 5.9: Variance of the spikes gradient at each layer on a feed-forward SNN with 20 layers with target rate of 30Hz. The input spikes were Poisson trains at the target rate. Each layer consisted of 2000 LIF neurons with parameters $\tau = 10\text{ms}$, $I = 0.9$, $V_r = 0$, $V_{th} = 1$. The optimal weights found had standard deviation $\sigma_w = 0.0096$. Optimisation was done using `scipy.root_scalar` function with `toms748` method. The surrogate function used was piecewise-constant as in (4.8) with $B_{th} = 0.01$, and $g = \kappa$ (corrected) or $g = 1$ (original). The network was simulated for 1s with timestep $\Delta t = 1\text{ms}$.

5.6 Discussion and Conclusions

The development of new techniques that allowed to train deeper and deeper ANNs has been an integral component of the success of ANNs in recent years. Proper weight initialisation has been shown to be crucial to achieve the best accuracy and

fast convergence on deep neural networks, especially when composed of many layers [241]. Empirical evidence has shown that compared to ANNs, deep SNNs are even more sensitive to suboptimal weight initialisation where networks with just a few layers already pose a challenge [240].

Although there is a wide range of weight initialisation methods for ANNs, relatively simple techniques based on variance propagation are the norm in most works. However, we have shown that this method is insufficient for the forward propagation of SNNs for two reasons. Firstly, the relationship between the variance of the membrane potential and the variance of the spikes (or equivalently the probability of spiking) is not straight forward. Secondly, we observed that the probability of spiking seems to decrease as we decrease the resolution of the SNN simulation (the firing rate collapse problem).

The first problem has already been tackled in the past by developing techniques based in finding the stationary membrane distribution. These include the diffusion approximation [24], shot-noise [248] or threshold integration [250]. These methods should be enough to allow to predict the output firing rate under a wide range of circumstances and neuron models. However, all of them are developed for ideal continuous-time simulations and do not consider the firing rate collapse problem.

The firing rate collapse problem has often been suggested to be just an inability of the neurons to fire enough times at low temporal resolutions. We have shown that this is not enough to fully explain this phenomenon, instead, the issue often arises from the fact that all inhibitory-excitatory inputs are cancelled instantaneously within a timestep. Thus, removing all output spikes that could take place at some permutation of the input spikes. We have shown that we can take this into account using different methods: random walk, Wiener noise or sampling a random permutation.

While these methods solve the firing rate collapse problem, it is yet to be seen what impact they can have in SNN training. Since the newly added spikes are

stochastic, this could lead to requiring more training steps to properly sample the network response. Yet, given that this is only important during initialisation, we could decrease the probability of firing as the training progresses and substitute this by a homeostatic rule as in Chapter 4 and other works such as [145, 240]. Moreover, we also need to develop a method to evaluate how many layers can we stack while somewhat accurately predicting the final rate when using real data instead of simple homogeneous Poisson spikes.

In addition, there is also the added computational cost which, in principle, should be small in the random walk and Wiener cases as the operations required are linear with the number of neurons ($O(n)$) and thus negligible compared to the synaptic cost ($O(n^2)$). For the permutation method, if properly implemented, the cost would be linear with the number of events. Since the number of events per timestep is bounded by the number of weights, this method would scale quadratically with the number of neurons ($O(n^2)$) and thus not increasing the overall computational complexity.

We also obtained an expression for the variance propagation of the gradients in the backward pass and empirically tested its effectiveness. Our results suggest that it is possible to ensure for the gradients to remain bounded by scaling the surrogate function provided some assumptions are met. While most of the assumptions are equivalent to those made on the ANN literature, the last assumption 5.13 is the most problematic one since it implies that backward surrogate gradients are uncorrelated. For this to be true we need to constrain the surrogate function to be very narrow. We showed in Chapter 4 that it is indeed possible to use a very narrow gradient while still learning efficiently which gives reason to believe that it is reasonable to use this assumption. Besides, our empirical results suggest that the gradients do indeed remain nearly constant when using this correction. However, empirical evidence is needed to ensure that this actually works with real data.

In this chapter we have assumed that the inputs are homogeneous Poisson spike trains. In real datasets this is rarely the case and in particular most interesting datasets are neither homogeneous nor Poisson. This is a very important issue that needs to be tackled for these methods to be successful. Yet, our results as well as previous work on SNN initialisation on real datasets are encouraging [240].

In brief, we have devised strategies to optimally initialise SNNs by considering their true membrane distribution and the problems arising from discrete time simulations of continuous systems. However, there is still the need to empirically prove their viability on real data.

Chapter 6

Conclusion

This final chapter reviews the contributions of this thesis within the current SNN training context and presents possible future research directions.

6.1 Future Work

We propose several future research directions to extend the work presented in this thesis and briefly review recent works that have built upon the research presented herein to some extent.

Neural Heterogeneity. In our work we studied the impact of adding heterogeneity of time constants in SNNs. Yet, there could be other neural mechanisms that could benefit from heterogeneity. Posterior works have shown that including heterogeneity in adaptive thresholds and using a multi-Gaussian surrogate function improves performance to a point where it is comparable to ANNs [258]. Other mechanisms have also been studied such as after-spike currents [259] and axonal delays [260], and suggest there are further advantages to heterogeneity such as robustness against correlated noise [179] and propagation speed of synaptic activity

[260]. There are also results that seem to point that adaptive mechanisms may not always be advantageous [261]. Nevertheless, there is still a lot of uncertainty as to when is heterogeneity useful, what desirable properties does it bring and what is the role and importance of each possible neural mechanism where heterogeneity can be introduced.

While these works (including ours) rely on artificial training mechanisms to tune heterogeneous parameters, we do not know how can these mechanisms are implemented, if at all, in a biological networks. Some recent work on the *Drosophila* fly's olfactory memory centre suggests that adding interneuronal variability degraded the memory performance but this could be recovered by adding compensatory mechanisms [262]. Yet, the mechanisms that take part in generating and regulating heterogeneity are still largely unknown.

Similarly, we found that time constants follow some distributions that are qualitatively similar to experimentally observed time constants. Some recent work has shown that the log-normal distributions of neuron densities seem to be ubiquitous across mammalian cerebral cortex [263]. It would be interesting to answer why this may be the case. It is possible that, similarly to how weight distributions of trained ANNs are Gaussian, there may be some mechanism that results in this kind of distributions across the brain.

We can also ask whether the lessons from heterogeneity can be applied to ANNs. Some works suggest improvements by mixing activation functions [264] and also that metalearning the activation functions leads to diversity in the activations [265]. There could also be other applications of heterogeneity beyond SNN or ANN training such as improving bio-inspired sensors such as spiking cameras [266].

Finally, we still do not have a theoretical justification to give an insight as to why heterogeneity may be a desirable feature for learning. Intuitively, it seems reasonable to use a dynamical system that behaves in diverse ways to learn from complex signals

but yet, to the best of our knowledge, there are no formal insights into why.

Sparse Spiking Gradient Descent. We mainly concentrated in accelerating the backward pass of an SNN, however, it would also be interesting to study the overall speedup we would obtain if we were to accelerate both forward and backward passes. Other types of sparsity can also be added such as weight sparsity as described in [234]. Moreover, the number of spikes can be restricted to a single one per neuron per input sample [267]. All of these changes could lead to yet even faster SNN implementations.

Another extension of this work could consist in adapting these methods to ANNs. For instance, very recent work suggests that the GRU architecture can be adapted to use the spiking non-linearity and achieve similar performance as their ANN counterparts while being moderately sparse in the forward and backward passes [268].

Finally, it is possible that there are other modifications at the GPU programming level that can speedup the inference and training process while not necessarily changing the network execution. Some recent examples include recomputing and timeskipping to enable for a longer number of timesteps [269].

Spiking Network Initialisation. This topic is still largely unexplored with very few references in the literature that have attempted to tackle it [240]. While the results we have presented are promising there are still several unresolved questions.

Firstly, there is a need to test whether the firing rate collapse solution is viable for training. This includes whether training actually takes place without issue and whether the newly added computations do not add too much computational overhead.

Secondly, a systematic analysis of different network architectures with different widths and depths is required to test to what extent the forward propagation method works for predicting the different layers firing rates. Importantly, this analysis needs

to also take into consideration what happens when we switch the input from homogeneous Poisson spike trains to real data.

Finally, we need to perform a similar analysis for the backward pass. This should include testing how well the different assumptions hold by measuring the variance of the gradients at different layers with real data and also, check whether this method is fast enough that it is reasonable to use it systematically for network initialisation.

6.2 Summary of Thesis Achievements

The goal of this thesis was to develop new techniques to advance the state of the art of deep spiking neural networks training. This work took place at a time where the revolutionary advances in the field of machine learning during the last decade started to become available for SNN training. Technical issues such as the non-differentiability of spikes hindered the adoption of deep learning techniques to train SNNs at scale until recently. Methods like surrogate gradient descent have started to become an easy and reliable way to train SNNs on large datasets with high accuracy and flexibility. Importantly, as opposed to previous SNN training methods, surrogate gradient learning combines several desirable features, namely it can be used to train large models on large datasets, it learns the spike times directly and it is able to create and remove spikes. These properties result in highly accurate models that can work with relatively few spikes thus enabling to harness the higher energy efficiency derived from the activation sparsity of SNNs. Despite all the desirable characteristics, training SNNs still suffers from important open questions and shortcomings that motivate this thesis.

Firstly, as opposed to most ANN architectures, SNNs are dynamical systems that attempt to model biological neurons in the brain. As such, SNN training can be used to study whether there are any advantages to using this kind of dynamical system for

learning as well as opening the possibility of gaining new insights into the brain. In Chapter 3, we attempted to answer whether the heterogeneity present in the brain has a functional advantage. We found that time constant heterogeneity improves learning on tasks with rich temporal structure and generalisation across data at different time scales. In other words, we confirmed an otherwise intuitive result: that heterogeneous dynamical systems with responses at different timescales tend to be better for processing temporally complex data. More unexpectedly, we also found that heterogeneity adds robustness against mistuned hyperparameters and that the learned time constants we obtained qualitatively matched time constants measured in biological neurons. Importantly, these results extend to neuromorphic computing systems. Heterogeneous initialisation is $O(1)$ and time constant training is $O(n)$ with the number of neurons. Meanwhile, adding more neurons results in a quadratic increase in the number of parameters to train. Altogether, these results suggest that heterogeneity is a compelling strategy for more effective and robust training for a relatively minor computational cost.

As we mentioned before, one of the main incentives for using SNNs is the potential to achieve the same level of power efficiency as the brain by leveraging the sparse activations of spiking neurons. However, while most effort has concentrated in efficiently emulating the forward pass of SNNs on specialised hardware, little has been done to improve the computational requirements of the backward pass. In Chapter 4, we attempted to do this by using surrogate functions that result in sparse gradients. We demonstrate that we can get the same performance as standard dense backpropagation at a much reduced computational and memory cost. To the best of our knowledge this was the first backpropagation method that leverages the sparse nature of spikes in training. We empirically showed that about 98% of the backward operations can be skipped and obtain up to 150x speedup and 85% memory savings. This results suggest that it is possible to achieve fast, energy and memory efficient end-to-end training on SNNs, which could have benefits for both

neuromorphic training and computational neuroscience research.

Finally, despite showing that SNNs can be successfully trained to solve difficult tasks and that training can indeed take place very efficiently, this can only be done provided the initial network parameters have been properly set. While often not well reflected in the literature, researchers frequently spend a lot of time fine tuning the precise weight initialisation in SNNs. In Chapter 5 we attempted to tackle this issue by using techniques from the ANN initialisation literature as well as computational neuroscience results. We saw that the problem of weight initialisation for ANNs is a much more nuanced problem than it is for ANNs due to the spike-and-reset non-linearity of SNNs and the firing rate collapse problem. On one hand, we found several solutions to the firing rate collapse problem under different sets of assumptions which successfully solve the issue by leveraging classical random walk and Wiener processes results. On the other hand we devised a general strategy for SNN initialisation which combines variance propagation techniques from ANNs and different methods to obtain the expected firing rate and membrane potential distribution based on diffusion and shot-noise approximations. Altogether, we obtained theoretical results to solve the SNN initialisation problem but it still remains to be seen whether this method can be successfully applied to SNNs, on real datasets and to what extent the results hold.

Overall, in this thesis we have tackled questions that aim to bring SNNs closer to the current ANNs status as universal function approximators in most machine learning tasks while at the same time bringing forward the opportunity to learn new insights from biological computation and achieve more efficient inference and training. We believe that the results and analysis presented here situate SNNs closer to this end.

Appendices

Appendix A

Gradient of an SNN layer

The unrolled expression of an SNN layer is given by

$$v^{(l+1)}[t+1] = \sum_{k=0}^t \alpha^{t-k} (W^{(l)} s^{(l)}[k] + (1 - \alpha) RI_{ext}) \quad (\text{A.1})$$

$$s_j^{(l+1)}[t+1] = H(v_j^{(l+1)}[t+1] - V_{th}) \quad (\text{A.2})$$

where $v^{(l+1)}[t+1], s^{(l+1)}[t+1] \in \mathbb{R}^{n^{(l+1)}}$, $W^{(l)} \in \mathbb{R}^{n^{(l+1)} \times n^{(l)}}$, $s^{(l)}[t+1] \in \mathbb{R}^{n^{(l)}}$, the rest of parameters are scalars and H is the Heaviside step function.

We assume that the loss function applied to the last layer $\varepsilon : \mathbb{R}^{n^{(L-1)}} \rightarrow \mathbb{R}$ is a function of the sum of potentials in the last layer across time (i.e. $\varepsilon(v^{(L-1)}[t]) = \mathcal{L}(\sum_t v^{(L-1)}[t])$). Thus, we can write

$$\frac{\partial \varepsilon}{\partial s^{(l)}[t]} = \sum_{k>t} \frac{\partial \varepsilon}{\partial v^{(l+1)}[k]} \frac{\partial v^{(l+1)}[k]}{\partial s^{(l)}[t]} \quad (\text{A.3})$$

where $\frac{\partial \varepsilon}{\partial s^{(l)}[t]} \in \mathbb{R}^{1 \times n^{(l)}}$, $\frac{\partial \varepsilon}{\partial v^{(l+1)}[k]} \in \mathbb{R}^{1 \times n^{(l+1)}}$ and $\frac{\partial v^{(l+1)}[k]}{\partial s^{(l)}[t]} \in \mathbb{R}^{n^{(l)} \times n^{(l+1)}}$ (using the numerator layout convention).

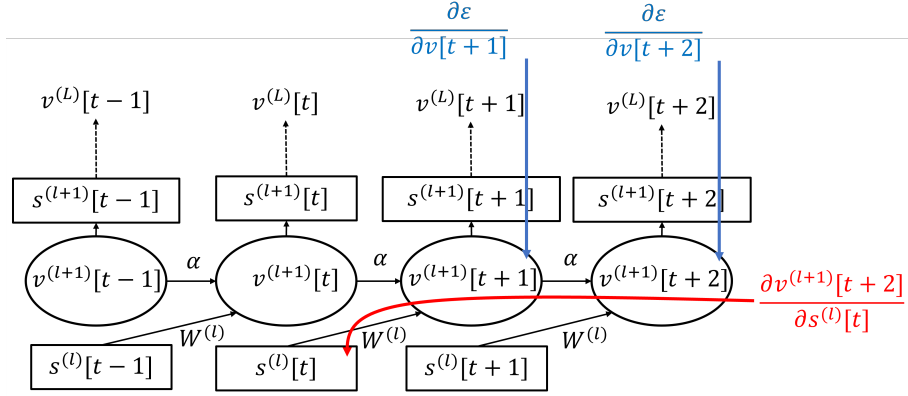


Figure A.1: Activation and gradient flow diagram for an unrolled SNN

Using equation (A.2) we can expand the first term in (A.3)

$$\left(\frac{\partial \varepsilon}{\partial v^{(l+1)}[t]} \right)_i = \left(\frac{\partial \varepsilon}{\partial s^{(l+1)}[t]} \right)_i H' \left(v_i^{(l+1)}[t] \right) \quad (\text{A.4})$$

Where H' is a function that subtracts the threshold V_{th} and then applies the *surrogate derivative* of the Heaviside function.

For the second term, we note that a spike at time t can only affect spikes in the future (check Figure A.1 for a visualisation), thus

$$\frac{\partial v^{(l+1)}[k]}{\partial s^{(l)}[t]} = \begin{cases} \frac{\partial v^{(l+1)}[t+1]}{\partial s^{(l)}[t]} \alpha^{k-t-1}, & k > t \\ 0, & \text{otherwise} \end{cases} \quad (\text{A.5})$$

we also have $\frac{\partial v^{(l+1)}[t+1]}{\partial s^{(l)}[t]} = \frac{\partial W^{(l)} s^{(l)}[t]}{\partial s^{(l)}[t]} = W^{(l)}$.

For notational clarity, we often use $\frac{\partial \varepsilon}{\partial z} := \Delta z$. Thus, after applying equations (A.5), equations (A.3) and (A.4) become

$$\Delta s^{(l)}[t] = \sum_{k>t} \alpha^{k-t-1} \Delta v^{(l+1)}[k] W^{(l)} \quad (\text{A.6})$$

$$\Delta v_i^{(l+1)}[t] = \Delta s_i^{(l+1)}[t] H' \left(v_i^{(l+1)}[t] \right) \quad (\text{A.7})$$

Appendix B

Supplementary Information on Sparse Spiking gradient descent

B.1 Training details

B.1.1 Conversion of Fashion-MNIST to spike latencies

We convert the normalised analogue grayscale pixel values x_i of the Fashion-MNIST samples [189] to spike times in a similar way to that used in [145]. Specifically, we convert the normalised intensities into spike times following

$$T(x) = \begin{cases} \tau_{\text{eff}} \log \left(\frac{x}{x-\theta} \right), & x > \theta \\ \infty, & \text{otherwise} \end{cases}$$

B.1.2 Spiking neuron model

We use the simplified Leaky Integrate and Fire model which is a variation on the standard Leaky Integrate and Fire model in which spikes are directly integrated

in the membrane without being filtered by beforehand. The membrane potential evolves according to

$$\tau \frac{dv_j^{(l+1)}(t)}{dt} = -(v_j^{(l+1)}(t) - V_0) + \sum_i s_i^{(l)}[t] W_{ij}^{(l)} \quad (\text{B.1})$$

The membrane potential resets to V_r as soon as it reaches a threshold V_{th} . The model keeps the main features of a spiking model, namely, a membrane potential that varies according to the incoming input spikes and synaptic weights, a leaky component that draws the potential towards V_{rest} and spiking and resetting mechanisms.

We use a discretised version of the model as in Equation (2.19) that we repeat here

$$v_j^{(l+1)}[t+1] = \alpha v_j^{(l+1)}[t] + (1 - \alpha)V_0 + \sum_i s_i^{(l)}[t] W_{ij}^{(l)} - (V_{th} - V_r)s_j^{(l+1)}[t] \quad (\text{B.2})$$

where we define $\alpha = \exp(-\Delta t/\tau)$. Spiking then takes place by applying $f(\cdot)$ on the membrane potential

$$s_j^{(l+1)}[t+1] = f(v_j^{(l+1)}[t+1]) \quad (\text{B.3})$$

which is defined as a unit step function centered at the threshold V_{th}

$$f(v) = \begin{cases} 1, & v \geq V_{th} \\ 0, & \text{otherwise} \end{cases} \quad (\text{B.4})$$

Thus, given an input spikes tensor $s^{(l)} \in \{0, 1\}^{B \times T \times n^{(l)}}$ (B being the batch size T number of time steps and $n^{(l)}$ number of neurons in layer l) and weights $W^{(l)} \in$

$\mathbb{R}^{n^{(l)} \times n^{(l+1)}}$ we obtain the output spikes $s^{(l+1)} \in \{0, 1\}^{B \times T \times n^{(l+1)}}$ by applying (B.2), (B.3), (B.4).

B.1.3 Network and weight initialisation

All layers are feed-forward fully connected in all experiments unless otherwise specified. The weights were sampled from a uniform distribution $\mathcal{U}(-\sqrt{N}, \sqrt{N})$ where N is the number of input connections to the layer. In all experiments we have an input layer, two hidden layers with the same number of neurons and a readout layer with as many neurons as output classes for the given dataset. The readout layer neurons are identical to the hidden layers except in the firing threshold which we set to infinity.

In the convolutional layer experiment, the first layer was a convolutional spiking layer with 64 filters, kernel size of 3 and stride of 1, followed by a max pool layer with kernel size of 2 and then flattened into a linear layer of 12544 inputs and 10 outputs.

B.1.4 Supervised and regularisation losses

We have a loss composed of three terms: cross entropy loss, higher activity regularisation and lower activity regularisation. The cross entropy loss is defined as

$$\mathcal{L}^{cross} = \frac{1}{B} \sum_{b=1}^B \sum_{c=1}^C y_{b,c} \log(p_{b,c})$$

With $y_{b,c} \in \{0, 1\}^C$ being a one hot target vector for batch sample b given a total batch size B and a total number of classes C . The probabilities $p_{b,c}$ are obtained using the Softmax function

$$p_{b,c} = \frac{e^{a_{b,c}}}{\sum_{c=1}^C e^{a_{b,c}}}$$

Here, the logits $a_{b,c} \in \mathbb{R}^{B \times C}$ are obtained by taking the max over the time dimension on the readout layer membrane potential $a_{b,c} = \max_t v_{b,c}^{(L-1)}[t]$. For the SHD dataset we used $a_{b,c} = \sum_t v_{b,c}^{(L-1)}[t]$

We also use two regularisation terms to constrain the spiking activity. These terms have been shown to not degrade the network performance [145]. First, we use a lower activity penalty to promote activity in the hidden layers.

$$\mathcal{L}_b^{low} = -\frac{\lambda^{low}}{n^{(l)}} \sum_i^{n^{(l)}} \left(\max(0, \nu^{low} - \zeta_{b,i}^{(l)}) \right)^2$$

Here $\zeta_{b,i}^{(l)} = \sum_t s_{b,i}^l[t]$ is the spike count of neuron i in layer l and batch sample b .

Secondly, the upper activity loss is added to prevent neurons to spike too often.

$$\mathcal{L}_b^{up} = -\lambda^{up} \max \left(0, \frac{1}{n^{(l)}} \sum_i^{n^{(l)}} \zeta_{b,i}^{(l)} - \nu^{up} \right)$$

Finally the overall loss is obtained

$$\mathcal{L} = \mathcal{L}^{cross} + \frac{1}{B} \sum_{b=1}^B (\mathcal{L}_b^{low} + \mathcal{L}_b^{up})$$

B.1.5 Surrogate gradient

We use the following function as the surrogate derivative of the firing function defined in (B.4).

$$\frac{df(x)}{dx} = g(x) = \frac{1}{(\beta|x - V_{th}| + 1)^2}$$

In the sparse case we use

$$\frac{df(x)}{dx} = \begin{cases} g(x), & \text{if } |x - V_{th}| < B_{th} \\ 0, & \text{otherwise} \end{cases} \quad (\text{B.5})$$

B.1.6 Parameters

Here we summarise all parameters used in each dataset.

	F-MNIST	N-MNIST	SHD
Number of Input Neurons	784	1156	700
Number of Hidden	200	200	200
Number of classes	10	10	20
Number epochs	100	100	200
B	256	256	256
T	100	300	500
Δt	1ms	1ms	2ms
τ hidden	10ms	10ms	10ms
τ readout	10ms	10ms	20ms
τ_{eff}	20ms	N/A	N/A
θ	0.2	N/A	N/A
V_r	0	0	0
V_0	0	0	0
V_{th}	1	1	1
B_{th}	0.2	0.2	0.2
β	100	100	100
Optimiser	Adam	Adam	Adam
Learning Rate	0.0002	0.0002	0.001
Betas	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)
$\lambda^{(low)}$	100	100	100
$\nu^{(low)}$	0.001	0.001	0.001
$\lambda^{(up)}$	0.06	0.06	0.06
$\nu^{(up)}$	1	1	10

Table B.1: Network and training parameters

B.2 Loss evolution for all datasets

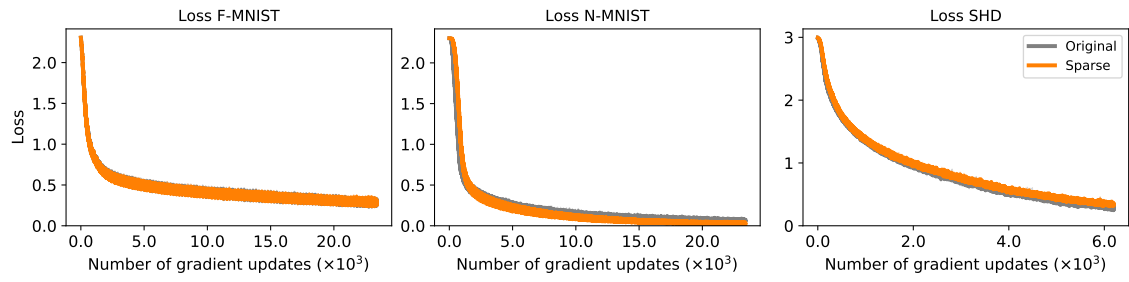


Figure B.1: Loss for each dataset when using 200 hidden neurons (5 samples)

Appendix C

Mean and variance of Poisson spikes filtered by the membrane

In a LIF neuron with instantaneous synapses with weight w_m distributed according to an arbitrary distribution $w_m \sim f_W(w)$, by time t the first spike input will have decayed to $w_m \cdot \exp(-\frac{t-t_1}{\tau})$, the second spike input will have decayed to $w_m \cdot \exp(-\frac{t-t_2}{\tau})$ and so on. Thus, the total input the neuron receives from synapse m in an interval of length t is given by

$$w_m \cdot s_m(t) = w_m \left[\exp\left(-\frac{t-t_1}{\tau}\right) + \exp\left(-\frac{t-t_2}{\tau}\right) + \dots \right] \quad (\text{C.1})$$

$$= w_m \sum_{i=1}^{N_m} \exp\left(-\frac{t-t_i}{\tau}\right) \quad (\text{C.2})$$

where the spike times $t_i \in [0, t]$. Then, the total input the neuron receives from all its synapses is given by

$$S(t) = \sum_{m=1}^M w_m s_m(t) \quad (\text{C.3})$$

We will concentrate now on the input from a single synapse (C.2) and will drop the subscript m for clarity.

Spikes arrive following a Homogeneous Poisson process with rate r . This means that the number of spikes N is a random variable and the probability that a total of $N=k$ spikes arrive in a time interval of length t is

$$P(N = k) = \frac{(rt)^k}{k!} e^{-rt} \quad (\text{C.4})$$

A direct consequence of this is that $\mathbb{E}[N] = rt$ (i.e. on average, we get rt spikes on an interval of length t) and the variance of the number of spikes is also $\text{Var}[N] = rt$.

In a Poisson process, arrival times conditioned to the number of arrivals N in the given interval t are independent and uniformly distributed, $t_i \sim U(0, t)$ [270]. We can obtain the conditional expectation of the input spikes given that N spikes arrived

$$\mathbb{E}[s(t|N)] = \mathbb{E}\left[\sum_{i=1}^N \exp\left(-\frac{t-t_i}{\tau}\right)\right] \quad (\text{C.5})$$

$$= \exp\left(-\frac{t}{\tau}\right) \sum_{i=1}^N \mathbb{E}\left[\exp\left(\frac{t_i}{\tau}\right)\right] \quad (\text{C.6})$$

$$= \exp\left(-\frac{t}{\tau}\right) \sum_{i=1}^N \int_0^t \frac{1}{t} \exp\left(\frac{t_i}{\tau}\right) dt_i \quad (\text{C.7})$$

$$= \exp\left(-\frac{t}{\tau}\right) \sum_{i=1}^N \frac{\tau}{t} \left[\exp\left(\frac{t}{\tau}\right) - 1\right] \quad (\text{C.8})$$

$$= \frac{\tau}{t} \sum_{i=1}^N \left[1 - \exp\left(-\frac{t}{\tau}\right)\right] \quad (\text{C.9})$$

$$= N \frac{\tau}{t} \left[1 - \exp\left(-\frac{t}{\tau}\right)\right] \quad (\text{C.10})$$

from this we can easily obtain $\mathbb{E}[s(t)]$ using the law of total expectation

$$\mathbb{E}[s(t)] = \sum_{k=0}^{\infty} \mathbb{E}[s(t|k)]P(N = k) \quad (\text{C.11})$$

$$= \sum_{k=0}^{\infty} k\mathbb{E}[s(t|1)]P(N = k) \quad (\text{C.12})$$

$$= \mathbb{E}[s(t|1)] \sum_{k=0}^{\infty} kP(N = k) \quad (\text{C.13})$$

$$= \mathbb{E}[s(t|1)]\mathbb{E}[N] \quad (\text{C.14})$$

$$= r\tau \left[1 - \exp\left(-\frac{t}{\tau}\right) \right] \quad (\text{C.15})$$

and finally since synapses have a weight w we can obtain the expected synaptic input

$$\mathbb{E}[w \cdot s(t)] = \mathbb{E}[w]\mathbb{E}[s(t)] \quad (\text{C.16})$$

$$= \mathbb{E}[w]r\tau \left[1 - \exp\left(-\frac{t}{\tau}\right) \right] \quad (\text{C.17})$$

We can also compute the conditional variance given N spikes happened in the interval

$$\text{Var}[s(t|N)] = \text{Var}\left[\sum_{i=1}^N \exp\left(-\frac{t-t_i}{\tau}\right)\right] \quad (\text{C.18})$$

$$= \exp\left(-\frac{2t}{\tau}\right) \text{Var}\left[\sum_{i=1}^N \exp\left(\frac{t_i}{\tau}\right)\right] \quad (\text{C.19})$$

$$= \exp\left(-\frac{2t}{\tau}\right) \sum_{i=1}^N \text{Var}\left[\exp\left(\frac{t_i}{\tau}\right)\right] \quad (\text{C.20})$$

we separately compute

$$\text{Var} \left[\exp \left(\frac{t_i}{\tau} \right) \right] = \mathbb{E} \left[\left(\exp \left(\frac{t_i}{\tau} \right) \right)^2 \right] - \left(\mathbb{E} \left[\exp \left(\frac{t_i}{\tau} \right) \right] \right)^2 \quad (\text{C.21})$$

$$= \frac{1}{t} \int_0^t \exp \left(\frac{2t_i}{\tau} \right) dt_i - \left(\frac{\tau}{t} \left[\exp \left(\frac{t}{\tau} \right) - 1 \right] \right)^2 \quad (\text{C.22})$$

$$= \frac{\tau}{2t} \left[\exp \left(\frac{2t}{\tau} \right) - 1 \right] - \left(\frac{\tau}{t} \left[\exp \left(\frac{t}{\tau} \right) - 1 \right] \right)^2 \quad (\text{C.23})$$

which then gives

$$\text{Var} [s(t|N)] = N \exp \left(-\frac{2t}{\tau} \right) \left(\frac{\tau}{2t} \left[\exp \left(\frac{2t}{\tau} \right) - 1 \right] - \left(\frac{\tau}{t} \left[\exp \left(\frac{t}{\tau} \right) - 1 \right] \right)^2 \right) \quad (\text{C.24})$$

and we obtain $\text{Var}[s(t)]$ using the law of total variance

$$\text{Var} [s(t)] = \mathbb{E}[\text{Var} [s(t|N)]] + \text{Var} [\mathbb{E}[s(t|N)]] \quad (\text{C.25})$$

$$= \mathbb{E}[N \text{Var} [s(t|1)]] + \text{Var} [N \mathbb{E}[s(t|1)]] \quad (\text{C.26})$$

$$= \text{Var} [s(t|1)] \mathbb{E}[N] + (\mathbb{E}[s(t|1)])^2 \text{Var} [N] \quad (\text{C.27})$$

$$= rt (\text{Var} [s(t|1)] + (\mathbb{E}[s(t|1)])^2) \quad (\text{C.28})$$

and finally the synaptic input variance

$$\text{Var}[w \cdot s(t)] = \text{Var}[w] \text{Var}[s(t)] + \text{Var}[w] (\mathbb{E}[s(t)])^2 + \text{Var}[s(t)] (\mathbb{E}[w])^2 \quad (\text{C.29})$$

where all the terms are known.

Returning to the total input expression which considers a total of M synapses and their corresponding weights w_m , and given that both weights and spike trains from

different synapses are independent, we have that

$$\mathbb{E}[S(t)] = \sum_{m=1}^M \mathbb{E}[w_m s_m(t)] \quad (\text{C.30})$$

and

$$\text{Var}[S(t)] = \sum_{m=1}^M \text{Var}[w_m s_m(t)] \quad (\text{C.31})$$

where now all terms are known.

These equations are empirically validated in the following Figure:

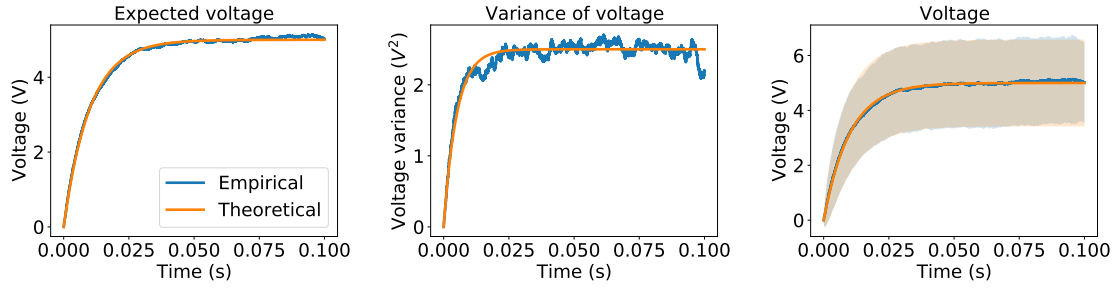


Figure C.1: Mean and variance evolution of the membrane potential as it is filtered through the membrane layer. Results averaged over 10 trials.

Finally, in the limit $t \rightarrow \infty$, we have

$$\lim_{t \rightarrow \infty} \mathbb{E}[S(t)] = \lim_{t \rightarrow \infty} \sum_{m=1}^M \mathbb{E}[w \cdot s(t)] \quad (\text{C.32})$$

$$= \lim_{t \rightarrow \infty} \sum_{m=1}^M \mathbb{E}[w] r \tau \left[1 - \exp\left(-\frac{t}{\tau}\right) \right] \quad (\text{C.33})$$

$$= \tau M r \mathbb{E}[w] \quad (\text{C.34})$$

If instead of a distribution of synapses we had K synaptic types we get as in (5.38)

$$\mu_v = \lim_{t \rightarrow \infty} \mathbb{E}[S(t)] = \tau \sum_k r_k w_k \quad (\text{C.35})$$

Next, taking the limit in (C.28) leads to

$$\lim_{t \rightarrow \infty} \text{Var}[s(t)] = \lim_{t \rightarrow \infty} r t \left(\text{Var}[s(t|1)] + (\mathbb{E}[s(t|1)])^2 \right) \quad (\text{C.36})$$

$$= \frac{\tau r}{2} \quad (\text{C.37})$$

Then,

$$\begin{aligned} \lim_{t \rightarrow \infty} \text{Var}[w \cdot s(t)] &= \\ &= \lim_{t \rightarrow \infty} \text{Var}[w] \text{Var}[s(t)] + \text{Var}[w] (\mathbb{E}[s(t)])^2 + \text{Var}[s(t)] (\mathbb{E}[w])^2 \end{aligned} \quad (\text{C.38})$$

$$= \frac{\tau r}{2} \text{Var}[w] + \text{Var}[w] (\tau r)^2 + \frac{\tau r}{2} (\mathbb{E}[w])^2 \quad (\text{C.39})$$

And again if instead of a distribution of synapses we had K synaptic types ($\text{Var}[w] = 0$ and $\mathbb{E}[w_k] = w_k$) we get a variance as in (5.38)

$$\sigma_v^2 = \lim_{t \rightarrow \infty} \text{Var}[S(t)] \quad (\text{C.40})$$

$$= \lim_{t \rightarrow \infty} \sum_k \text{Var}[w_k \cdot s(t)] \quad (\text{C.41})$$

$$= \sum_k \frac{\tau r_k}{2} (\mathbb{E}[w_k])^2 \quad (\text{C.42})$$

$$= \frac{\tau}{2} \sum_k r_k w_k^2 \quad (\text{C.43})$$

Appendix D

Threshold Integration in the shot-noise case

This Appendix follows the derivations in the supplementary material in [\[248\]](#).

Similarly to the diffusion case, we can numerically find the membrane distribution and firing rate for the shot noise equations. In this case, however, we start with three simultaneous differential equations corresponding to the continuity equation and the excitatory and inhibitory fluxes

$$\begin{aligned}\frac{\partial J(v, t)}{\partial v} &= -\frac{\partial P(v, t)}{\partial t} + r_{out}(t) (\delta(v - V_r) - \delta(v - V_{th})) \\ \frac{\partial J_e(v, t)}{\partial v} &= -\frac{J_e(v, t)}{w_e} + r_e P(v, t) - r_{out}(t) \delta(v - V_{th}) \\ \frac{\partial J_i(v, t)}{\partial v} &= -\frac{J_i(v, t)}{w_i} + r_i P(v, t)\end{aligned}$$

Note that we have three equations and four unknowns, however, we can rearrange [\(5.44\)](#) to obtain $P(v, t) = \frac{\tau}{v} (J_e(v, t) + J_i(v, t) - J(v, t))$ which in the steady state leads to

$$\begin{aligned}
\frac{\partial J(v)}{\partial v} &= r_{out}(t) (\delta(v - V_r) - \delta(v - V_{th})) \\
\frac{\partial J_e(v)}{\partial v} &= -\frac{J_e(v)}{w_e} + \frac{r_e \tau}{v} (J_e(v) + J_i(v) - J(v)) - r_{out}(t) \delta(v - V_{th}) \\
\frac{\partial J_i(v)}{\partial v} &= -\frac{J_i(v)}{w_i} + \frac{r_i \tau}{v} (J_e(v) + J_i(v) - J(v)) + J_i(V_{lb}) \delta(v - V_{lb}) \quad (D.1)
\end{aligned}$$

Note that we added the term $J_i(V_{lb})\delta(v - V_{lb})$ to ensure that the voltage is fully contained between V_{lb} and V_{th} , however, given V_{lb} is sufficiently low the effect of this is negligible.

In principle, it seems we should be able to integrate these equations as we did before, however, the v in the denominator is problematic as we approach $v=0$.

This can be solve by splitting the integration into two parts. Firstly, we integrate backwards from V_{th} to 0. Secondly we integrate forward from V_{lb} to 0. This leads to the need of adding the initial condition $P(V_{lb})=0$ which should be okay as long as V_{lb} is low enough.

Then, for the first integration we consider

$$\begin{aligned}
J(v) &= r_{out} f(v) \\
J_e(v) &= r_{out} f_e(v) \\
J_i(v) &= r_{out} f_i(v) \quad (D.2)
\end{aligned}$$

and equations (D.1) become

$$\begin{aligned}
\frac{\partial f(v)}{\partial v} &= \delta(v - V_r) \\
\frac{\partial f_e(v)}{\partial v} &= -\frac{f_e(v)}{w_e} + \frac{r_e \tau}{v} (f_e(v) + f_i(v) - f(v)) \\
\frac{\partial f_i(v)}{\partial v} &= -\frac{f_i(v)}{w_i} + \frac{r_i \tau}{v} (f_e(v) + f_i(v) - f(v))
\end{aligned} \tag{D.3}$$

and we start integrating with initial conditions $f(V_{th}) = f_e(V_{th}) = 1$ and $f_i(V_{th}) = 1$

For the second integration we consider

$$\begin{aligned}
J(v) &= |J_i(v)|g(v) \\
J_e(v) &= |J_i(v)|g_e(v) \\
J_i(v) &= |J_i(v)|g_i(v)
\end{aligned} \tag{D.4}$$

and equations (D.1) become

$$\begin{aligned}
\frac{\partial g(v)}{\partial v} &= 0 \\
\frac{\partial g_e(v)}{\partial v} &= -\frac{g_e(v)}{w_e} + \frac{r_e \tau}{v} (g_e(v) + g_i(v) - g(v)) \\
\frac{\partial g_i(v)}{\partial v} &= -\frac{g_i(v)}{w_i} + \frac{r_i \tau}{v} (g_e(v) + g_i(v) - g(v))
\end{aligned} \tag{D.5}$$

with initial conditions $g(V_{lb}) = g_e(V_{lb}) = 0$ and $g_i(V_{lb}) = 1$

After solving (D.3) and (D.5) we obtain the firing rate from the fact that $P(v)$ must integrate to 1 and that at all three functions must be continuous at 0, i.e.

$|J_i(v)|g_e(0-) = r_{out}f_e(0+)$. This leads to

$$|J_i(v)|/r_{out} = f_e(0+)/g_e(0-)$$

Then we can integrate to obtain the firing rate and reconstruct the fluxes and probability distribution

$$\frac{1}{r_{out}} = \frac{1}{r_{out}} \int_{V_{lb}}^{V_{th}} P(v) dv = \int_{V_{lb}}^0 \frac{|J_i(v)|}{r_{out}} \frac{g_e(v) + g_i(v) - g(v)}{v} dv + \int_0^{V_{th}} \frac{f_e(v) + f_i(v) - f(v)}{v} dv$$

So far, we assumed that $V_r > 0$ by placing the $\delta(v - V_r)$ term in the first equation in (D.3), yet if we had the case $V_r < 0$ we should place it in the first equation in (D.5). We could also add an external current by simply substituting v by $v - RI_{ext}$ and integrating from V_{th} to RI_{ext} and V_{lb} to RI_{ext} . We also note that we only consider cases where $RI_{ext} < V_{th}$.

Appendix E

Supplementary results on Firing rate collapse correction

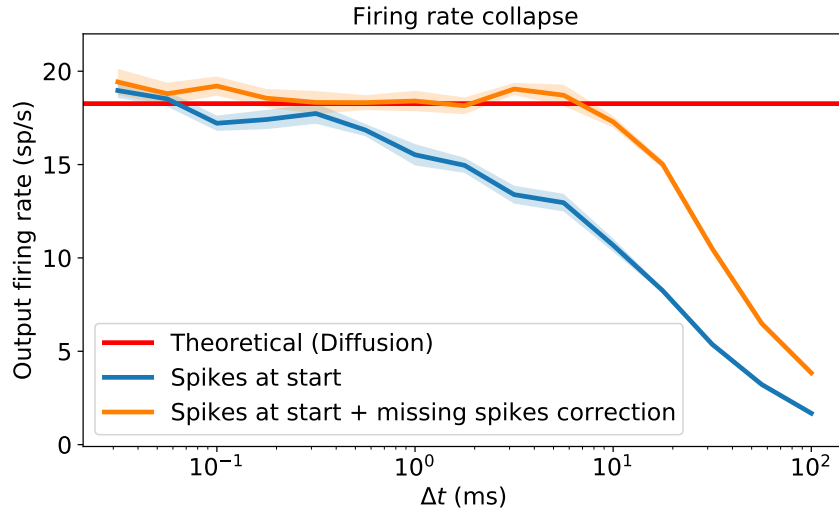


Figure E.1: Firing rate collapse as the simulation timestep increases and correction. Simulation details are identical to Figure 5.2. The integration is performed identically but neurons that did not spike but could have spiked are made to spike randomly according to Theorem 5.1 but now we use $\bar{\alpha}$ for the computation of y . Failure after $\Delta t = 20ms$ is due to the 50Hz Poisson generators no longer able to keep up.

Appendix F

Proofs on Vanishing-Exploding gradients on SNNs

Theorem 5.5. *Given assumptions 5.4, 5.5, 5.6 and 5.7 are satisfied and further assuming that the membrane potential distribution is symmetric around zero, the probability that a neuron fires within a timestep is bounded according to*

$$\rho^{(l+1)} < \frac{n^{(l)} \text{Var} [w_{ij}^{(l)}]}{2V_{th}^2} \left(A(\alpha) \rho^{(l)} + B(\alpha) (\rho^{(l)})^2 \right) \quad (5.91)$$

with $A(\alpha) = \frac{1}{1-\alpha^2}$, $B(\alpha) = \frac{2\alpha}{(1-\alpha)^2(1+\alpha)}$.

Proof.

We want to compute $\text{Var} [v_j^{(l+1)}[t+1]]$ as a function of parameters of the previous layer. We begin with

$$\begin{aligned}
Var \left[v_j^{(l+1)}[t+1] \right] &= Var \left[\sum_{k=0}^t \alpha^{t-k} \left(\sum_i w_{ij}^{(l)} s_i^{(l)}[k] + (1-\alpha) RI_{ext} \right) \right] \\
&= Var \left[\sum_{k=0}^t \alpha^{t-k} \left(\sum_i w_{ij}^{(l)} s_i^{(l)}[k] \right) \right] \\
&= Var \left[\sum_i w_{ij}^{(l)} \sum_{k=0}^t \alpha^{t-k} s_i^{(l)}[k] \right] \\
&= Var \left[\sum_i w_{ij}^{(l)} c_i^{(l)}[t] \right] \\
&= n^{(l)} Var \left[w_{ij}^{(l)} \right] E \left[\left(c_i^{(l)}[t] \right)^2 \right] \tag{F.1}
\end{aligned}$$

Since we assumed spikes times are independent (Assumption 5.7) we have that

$Cov \left[s_i^{(l)}[k], s_i^{(l)}[p] \right] = Var \left[s_i^{(l)}[k] \right] \delta_{pk}$ and we can expand

$$\begin{aligned}
E \left[\left(c_i^{(l)}[t] \right)^2 \right] &= E \left[\left(\sum_{k=0}^t \alpha^{t-k} s_i^{(l)}[k] \right)^2 \right] \\
&= E \left[\sum_{k=0}^t \sum_{p=0}^t \alpha^{t-k} \alpha^{t-p} s_i^{(l)}[k] s_i^{(l)}[p] \right] \\
&= \sum_{k=0}^t \sum_{p=0}^t \alpha^{t-k} \alpha^{t-p} E \left[s_i^{(l)}[k] s_i^{(l)}[p] \right] \\
&= \sum_{k=0}^t \sum_{p=0}^t \alpha^{t-k} \alpha^{t-p} Cov \left[s_i^{(l)}[k], s_i^{(l)}[p] \right] + \sum_{k=0}^t \sum_{p=0}^t \alpha^{t-k} \alpha^{t-p} E \left[s_i^{(l)}[k] \right] E \left[s_i^{(l)}[p] \right] \\
&= \sum_{k=0}^t \left(\alpha^{t-k} \right)^2 Var \left[s_i^{(l)}[k] \right] + \left(\sum_{k=0}^t \alpha^{t-k} E \left[s_i^{(l)}[k] \right] \right)^2 \tag{F.2}
\end{aligned}$$

Then, as we assumed spike times are independent and spikes occur with probability

$\rho^{(l)}$ (Assumption 5.8) we have that $E[s^q] = \int (1-\rho^{(l)}) s^q \delta(s) ds + \int \rho^{(l)} s^q \delta(s-1) ds =$

$\rho^{(l)}$. Substituting in (F.2)

$$\begin{aligned}
E \left[\left(c_i^{(l)}[t] \right)^2 \right] &= \left(\rho^{(l)} - (\rho^{(l)})^2 \right) \sum_{k=0}^t (\alpha^{t-k})^2 + (\rho^{(l)})^2 \left(\sum_{k=0}^t \alpha^{t-k} \right)^2 \\
&= \left(\rho^{(l)} - (\rho^{(l)})^2 \right) \frac{1 - (\alpha^2)^{t+1}}{1 - \alpha^2} + (\rho^{(l)})^2 \left(\frac{1 - \alpha^{t+1}}{1 - \alpha} \right)^2 \\
&\approx \left(\rho^{(l)} - (\rho^{(l)})^2 \right) \frac{1}{1 - \alpha^2} + (\rho^{(l)})^2 \left(\frac{1}{1 - \alpha} \right)^2 \tag{F.3}
\end{aligned}$$

where the last approximation holds for large enough t . For instance, if $\Delta t = 1\text{ms}$ and $\tau = 10\text{ms}$ after 50 timesteps the relative error of the approximation is lower than 1.5% for both sums.

With this result we obtain a final expression by substituting (F.3) back in (F.1)

$$\begin{aligned}
\text{Var} \left[v_j^{(l+1)}[t+1] \right] &\approx n^{(l)} \text{Var} \left[w_{ij}^{(l)} \right] \left(\left(\rho^{(l)} - (\rho^{(l)})^2 \right) \frac{1}{1 - \alpha^2} + (\rho^{(l)})^2 \left(\frac{1}{1 - \alpha} \right)^2 \right) \\
&= n^{(l)} \text{Var} \left[w_{ij}^{(l)} \right] \left(\frac{1}{1 - \alpha^2} \rho^{(l)} + \frac{2\alpha}{(1 - \alpha)^2(1 + \alpha)} (\rho^{(l)})^2 \right) \\
&= n^{(l)} \text{Var} \left[w_{ij}^{(l)} \right] \left(A(\alpha) \rho^{(l)} + B(\alpha) (\rho^{(l)})^2 \right) \tag{F.4}
\end{aligned}$$

Using Chebyshev's inequality and the fact that we assumed the membrane potential is symmetric around zero we can write

$$\begin{aligned}
P(|v_j^{(l+1)}[t+1]| \geq V_{th}) &= 2P(v_j^{(l+1)}[t+1] \geq V_{th}) \\
&< \frac{\text{Var} \left[v_j^{(l+1)}[t+1] \right]}{V_{th}^2} \tag{F.5}
\end{aligned}$$

Since $\rho^{(l+1)} = P(v_j^{(l+1)}[t+1] \geq V_{th})$ we have

$$\rho^{(l+1)} < \frac{\text{Var} \left[v_j^{(l+1)}[t+1] \right]}{2V_{th}^2} \tag{F.6}$$

Using this on (F.4) we finally obtain

$$\rho^{(l+1)} < \frac{n^{(l)} \text{Var} [w_{ij}^{(l)}]}{2V_{th}^2} \left(A(\alpha) \rho^{(l)} + B(\alpha) (\rho^{(l)})^2 \right) \quad (\text{F.7})$$

□

Theorem 5.6. *Provided assumptions 5.10 through 5.11 are satisfied, the variance of the gradient of the loss function with respect to the spikes is given by*

$$\text{Var} [\Delta s_j^{(l)}[t]] = \frac{\rho_a^{(l+1)}}{1 - \alpha^2} n^{(l+1)} \text{Var} [w_{ij}^{(l)}] \text{Var} [\Delta s_i^{(l+1)}[t]] \quad (5.94)$$

$$\text{where } \rho_a^{(l)} = E \left[\left(H' \left(v_i^{(l+1)}[t] \right) \right)^2 \right].$$

Proof.

We begin by expanding using Assumptions 5.10 and 5.9

$$\begin{aligned} \text{Var} [\Delta s_j^{(l)}[t]] &= \text{Var} \left[\sum_i w_{ij}^{(l)} \sum_{k>t} \alpha^{k-t-1} \Delta v_i^{(l+1)}[k] \right] \\ &= n^{(l+1)} \text{Var} \left[w_{ij}^{(l)} \sum_{k>t} \alpha^{k-t-1} \Delta v_i^{(l+1)}[k] \right] \\ &= n^{(l+1)} E [w_{ij}^{(l)2}] E \left[\left(\sum_{k>t} \alpha^{k-t-1} \Delta v_i^{(l+1)}[k] \right)^2 \right] \end{aligned} \quad (\text{F.8})$$

We further expand the last term using Assumptions 5.12 and 5.13

$$\begin{aligned}
E \left[\left(\sum_{k>t} \alpha^{k-t-1} \Delta v_i^{(l+1)}[k] \right)^2 \right] &= E \left[\sum_{k>t} \sum_{p>t} \alpha^{k-t-1} \alpha^{p-t-1} \Delta v_i^{(l+1)}[k] \Delta v_i^{(l+1)}[p] \right] \\
&= \sum_{k>t} \sum_{p>t} \alpha^{k-t-1} \alpha^{p-t-1} E \left[\Delta v_i^{(l+1)}[k] \Delta v_i^{(l+1)}[p] \right] \\
&= \sum_{k>t} \sum_{p>t} \alpha^{k-t-1} \alpha^{p-t-1} Cov \left[\Delta v_i^{(l+1)}[k], \Delta v_i^{(l+1)}[p] \right] \\
&\quad + \left(\sum_{k>t} \alpha^{k-t-1} E \left[\Delta v_i^{(l+1)}[p] \right] \right)^2 \\
&= \sum_{k>t} (\alpha^{k-t-1})^2 Var \left[\Delta v_i^{(l+1)}[k] \right] + \left(\sum_{k>t} \alpha^{k-t-1} E \left[\Delta v_i^{(l+1)}[p] \right] \right)^2
\end{aligned} \tag{F.9}$$

Substituting (F.9) back into (F.8)

$$\begin{aligned}
Var \left[\Delta s_j^{(l)}[t] \right] &= n^{(l+1)} E \left[w_{ij}^{(l)^2} \right] \left(\sum_{k>t} (\alpha^{k-t-1})^2 Var \left[\Delta v_i^{(l+1)}[k] \right] + \left(\sum_{k>t} \alpha^{k-t-1} E \left[\Delta v_i^{(l+1)}[p] \right] \right)^2 \right) \\
&\approx n^{(l+1)} E \left[w_{ij}^{(l)^2} \right] \left(\frac{1}{1-\alpha^2} Var \left[\Delta v_i^{(l+1)}[k] \right] + \frac{1}{(1-\alpha)^2} \left(E \left[\Delta v_i^{(l+1)}[p] \right] \right)^2 \right)
\end{aligned} \tag{F.10}$$

where the last equality holds as $k-t$ is increases.

Now we need to compute the terms $Var \left[\Delta v_i^{(l+1)}[k] \right]$ and $E \left[\Delta v_i^{(l+1)}[p] \right]$. For this, the following result will be useful

$$\begin{aligned}
E \left[\Delta s_j^{(l+1)}[t] \right] &= E \left[\sum_i w_{ij}^{(l)} \sum_{k>t} \alpha^{k-t-1} \Delta v_i^{(l+1)}[k] \right] \\
&= n^{(l+1)} E \left[w_{ij}^{(l)} \right] E \left[\sum_{k>t} \alpha^{k-t-1} \Delta v_i^{(l+1)}[k] \right] \\
&= 0
\end{aligned} \tag{F.11}$$

Then using Assumption 5.11 we have

$$\begin{aligned} E \left[\Delta v_i^{(l+1)}[t] \right] &= E \left[\Delta s_i^{(l+1)}[t] H' \left(v_i^{(l+1)}[t] \right) \right] \\ &= E \left[\Delta s_i^{(l+1)}[t] \right] E \left[H' \left(v_i^{(l+1)}[t] \right) \right] \end{aligned} \quad (\text{F.12})$$

$$= 0 \quad (\text{F.13})$$

and also

$$\begin{aligned} \text{Var} \left[\Delta v_i^{(l+1)}[t] \right] &= \text{Var} \left[\Delta s_i^{(l+1)}[t] H' \left(v_i^{(l+1)}[t] \right) \right] \\ &= E \left[\left(\Delta s_i^{(l+1)}[t] \right)^2 \right] E \left[\left(H' \left(v_i^{(l+1)}[t] \right) \right)^2 \right] - E \left[\left(\Delta s_i^{(l+1)}[t] \right) \right]^2 E \left[H' \left(v_i^{(l+1)}[t] \right) \right]^2 \\ &= E \left[\left(\Delta s_i^{(l+1)}[t] \right)^2 \right] E \left[\left(H' \left(v_i^{(l+1)}[t] \right) \right)^2 \right] \\ &= E \left[\left(\Delta s_i^{(l+1)}[t] \right)^2 \right] \rho_a^{(l+1)} \end{aligned} \quad (\text{F.14})$$

And finally substituting (F.12) and (F.14) into (F.10) we obtain

$$\begin{aligned} \text{Var} \left[\Delta s_j^{(l)} \right] &\approx n^{(l+1)} E \left[w_{ij}^{(l)^2} \right] \left(\frac{1}{1 - \alpha^2} \text{Var} \left[\Delta v_i^{(l+1)}[k] \right] + \frac{1}{(1 - \alpha)^2} \left(E \left[\Delta v_i^{(l+1)}[p] \right] \right)^2 \right) \\ &= n^{(l+1)} E \left[w_{ij}^{(l)^2} \right] \frac{1}{1 - \alpha^2} E \left[\left(\Delta s_i^{(l+1)}[t] \right)^2 \right] \rho_a^{(l+1)} \\ &= \frac{\rho_a^{(l+1)}}{1 - \alpha^2} n^{(l+1)} \text{Var} \left[w_{ij}^{(l)} \right] \text{Var} \left[\Delta s_i^{(l+1)}[t] \right] \end{aligned} \quad (\text{F.15})$$

□

Appendix G

Network Initialisation limit cases

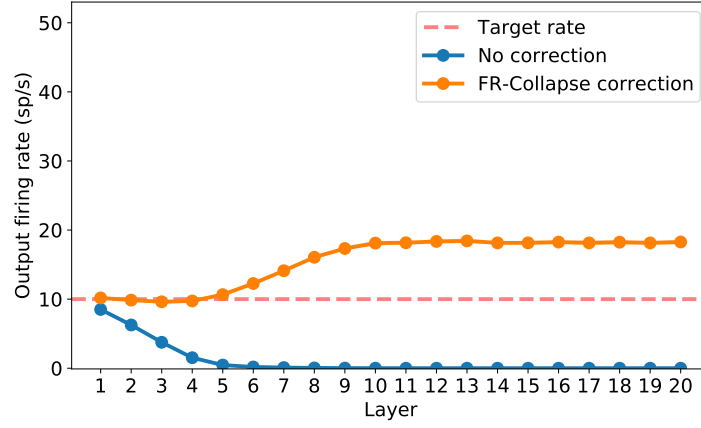


Figure G.1: Firing rate at each layer on a feed-forward SNN with 20 layers with target rate 10Hz. Simulations details are identical to Figure 5.8 but now with a different target rate and input rate (10Hz). The optimal weight was 0.030

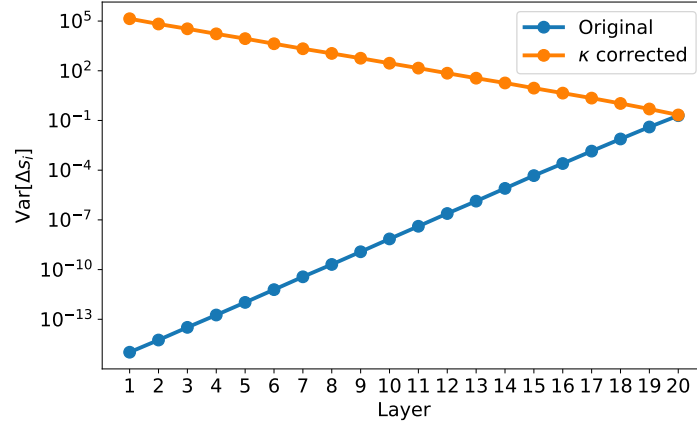


Figure G.2: Variance of the spike gradient at each layer on a feed-forward SNN with 20 layers and $B_{th}=0.1$. Other simulations details are identical to Figure 5.9.

Bibliography

- [1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [2] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.
- [3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [4] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451, Florence, Italy, July 2019. Association for Computational Linguistics.
- [5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

- [6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [7] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
- [8] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
- [9] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Židek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.
- [10] Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. *Advances in neural information processing systems*, 31, 2018.
- [11] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.
- [12] Timo C Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):1–17, 2021.

- [13] S Ramón y Cajal. Neuron theory or reticular theory? objective evidence of the anatomical unity of nerve cells. *English translation by M. Ubeda Purkiss and CA Fox, Consejo Superior de Investigaciones Científicas, Madrid, 1954.*
- [14] Yu Mu, Davis V Bennett, Mikail Rubinov, Sujatha Narayan, Chao-Tsung Yang, Masashi Tanimoto, Brett D Mensh, Loren L Looger, and Misha B Ahrens. Glia accumulate evidence that actions are futile and suppress unsuccessful behavior. *Cell*, 178(1):27–43, 2019.
- [15] Jun Nagai, Xinzhu Yu, Thomas Papouin, Eunji Cheong, Marc R Freeman, Kelly R Monk, Michael H Hastings, Philip G Haydon, David Rowitch, Shai Shaham, et al. Behaviorally consequential astrocytic regulation of neural circuits. *Neuron*, 109(4):576–596, 2021.
- [16] Paulo Kofuji and Alfonso Araque. Astrocytes and behavior. *Annual review of neuroscience*, 44:49–67, 2021.
- [17] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.
- [18] David A Drachman. Do we have brain to spare?, 2005.
- [19] Jonathan W Mink, Robert J Blumenshine, and David B Adams. Ratio of central nervous system to body metabolism in vertebrates: its constancy and functional basis. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, 241(3):R203–R212, 1981.
- [20] Shun’ichi Amari et al. *The handbook of brain theory and neural networks*. MIT press, 2003.

- [21] Blausen Medical. Medical gallery of blausen medical 2014. *WikiJournal of Medicine*, 1(2):1–79, 2014.
- [22] BruceBlaus. Neural tissue, 2013. Online. Accessed December 18, 2022 at https://commons.wikimedia.org/wiki/File:Blausen_0672_NeuralTissue.png.
- [23] Laurence F Abbott and Peter Dayan. Theoretical neuroscience. *Comput Math Model Neural*, 60:489–95, 2001.
- [24] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.
- [25] William M Connelly, Vincenzo Crunelli, and Adam C Errington. Variable action potential backpropagation during tonic firing and low-threshold spike bursts in thalamocortical but not thalamic reticular nucleus neurons. *Journal of Neuroscience*, 37(21):5319–5333, 2017.
- [26] William M Connelly, Vincenzo Crunelli, and Adam C Errington. Action potential, 2021. Online. Accessed December 18, 2022 at https://commons.wikimedia.org/wiki/File:Action_potential8.jpg.
- [27] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.
- [28] Rodolfo R Llinás. The intrinsic electrophysiological properties of mammalian neurons: insights into central nervous system function. *Science*, 242(4886):1654–1664, 1988.
- [29] Bertil Hille and Bertil Hille. *Ionic channels of excitable membranes*, volume 21. Springer, 1992.

- [30] Mitsuo Tanabe, Beat H Gähwiler, and Urs Gerber. L-type Ca^{2+} channels mediate the slow Ca^{2+} -dependent afterhyperpolarization current in rat Ca^{3+} pyramidal cells in vitro. *Journal of neurophysiology*, 80(5):2268–2273, 1998.
- [31] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.
- [32] Khashayar Pakdaman, Michele Thieullen, and Gilles Wainrib. Fluid limit theorems for stochastic hybrid systems with application to neuron models. *Advances in Applied Probability*, 42(3):761–794, 2010.
- [33] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.
- [34] Jinichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 1962.
- [35] Catherine Morris and Harold Lecar. Voltage oscillations in the barnacle giant muscle fiber. *Biophysical journal*, 35(1):193–213, 1981.
- [36] James L Hindmarsh and RM Rose. A model of neuronal bursting using three coupled first order differential equations. *Proceedings of the Royal society of London. Series B. Biological sciences*, 221(1222):87–102, 1984.
- [37] Louis Lapique. Recherches quantitatives sur l’excitation électrique des nerfs traitée comme une polarisation. *Journal of Physiology and Pathology*, 9:620–635, 1907.
- [38] Nicolas Fourcaud-Trocmé, David Hansel, Carl Van Vreeswijk, and Nicolas Brunel. How spike generation mechanisms determine the neuronal response to fluctuating inputs. *Journal of neuroscience*, 23(37):11628–11640, 2003.

- [39] Wulfram Gerstner, Raphael Ritz, and J Leo Van Hemmen. Why spikes? hebbian learning and retrieval of time-resolved excitation patterns. *Biological cybernetics*, 69(5):503–515, 1993.
- [40] Wulfram Gerstner, J Leo Van Hemmen, and Jack D Cowan. What matters in neuronal locking? *Neural computation*, 8(8):1653–1676, 1996.
- [41] Brian N Lundstrom, Matthew H Higgs, William J Spain, and Adrienne L Fairhall. Fractional differentiation by neocortical pyramidal neurons. *Nature neuroscience*, 11(11):1335–1342, 2008.
- [42] Amit Manwani and Christof Koch. Detecting and estimating signals in noisy cable structures, i: Neuronal noise sources. *Neural computation*, 11(8):1797–1829, 1999.
- [43] Elad Schneidman, Barry Freedman, and Idan Segev. Ion channel stochasticity may be critical in determining the reliability and precision of spike timing. *Neural computation*, 10(7):1679–1703, 1998.
- [44] Neal A Hessler, Aneil M Shirke, and Roberto Malinow. The probability of transmitter release at a mammalian central synapse. *Nature*, 366(6455):569–572, 1993.
- [45] Carl Van Vreeswijk and Haim Sompolinsky. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, 274(5293):1724–1726, 1996.
- [46] Srdjan Ostojic. Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons. *Nature neuroscience*, 17(4):594–600, 2014.
- [47] Omri Harish and David Hansel. Asynchronous rate chaos in spiking neuronal circuits. *PLoS computational biology*, 11(7):e1004266, 2015.

- [48] Peter E Kloeden and Eckhard Platen. Stochastic differential equations. In *Numerical solution of stochastic differential equations*, pages 103–160. Springer, 1992.
- [49] Daniel T Gillespie. The mathematics of brownian motion and johnson noise. *American Journal of Physics*, 64(3):225–240, 1996.
- [50] Richard B Stein. A theoretical analysis of neuronal variability. *Biophysical Journal*, 5(2):173–194, 1965.
- [51] Richard B Stein. Some models of neuronal variability. *Biophysical journal*, 7(1):37–68, 1967.
- [52] John A Gubner. *Probability and random processes for electrical and computer engineers*, page 338;411. Cambridge University Press, 2006.
- [53] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [54] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [55] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [56] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.
- [57] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

- [58] Gustaf Gripenberg. Approximation by neural networks with a bounded number of nodes at each level. *Journal of approximation theory*, 122(2):260–266, 2003.
- [59] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.
- [60] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.
- [61] Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.
- [62] Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.
- [63] Sejun Park, Chulhee Yun, Jaeho Lee, and Jinwoo Shin. Minimum width for universal approximation. *arXiv preprint arXiv:2006.08859*, 2020.
- [64] Vitaly Maiorov and Allan Pinkus. Lower bounds for approximation by mlp neural networks. *Neurocomputing*, 25(1-3):81–91, 1999.
- [65] Namig J Guliyev and Vugar E Ismailov. Approximation capability of two hidden layer feedforward neural networks with fixed weights. *Neurocomputing*, 316:262–269, 2018.
- [66] Namig J Guliyev and Vugar E Ismailov. On the approximation by single hidden layer feedforward neural networks with fixed weights. *Neural Networks*, 98:296–304, 2018.
- [67] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

- [68] Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794, 2020.
- [69] Wolfgang Maass. Fast sigmoidal networks via spiking neurons. *Neural computation*, 9(2):279–304, 1997.
- [70] Wolfgang Maass. Computing with spiking neurons. *Pulsed neural networks*, 2:55–85, 1999. <https://igi-web.tugraz.at/PDF/100.pdf>.
- [71] Iulia M Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8529–8533. IEEE, 2020.
- [72] Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176. Springer, 2011.
- [73] Donald Olding Hebb. *The Organization of Behavior: A Psychological Theory*. Wiley New York, 1949.
- [74] Carla J Shatz. The developing brain. *Scientific American*, 267(3):60–67, 1992.
- [75] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.
- [76] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.
- [77] Kenneth D Miller, Joseph B Keller, and Michael P Stryker. Ocular dominance column development: analysis and simulation. *Science*, 245(4918):605–615, 1989.

- [78] Ralph Linsker. From basic network principles to neural architecture: emergence of orientation columns. *Proceedings of the National Academy of Sciences*, 83(22):8779–8783, 1986.
- [79] Kenneth D Miller. A model for the development of simple cell receptive fields and the ordered arrangement of orientation columns through activity-dependent competition between on-and off-center inputs. *Journal of Neuroscience*, 14(1):409–441, 1994.
- [80] Philippe G Schyns. A modular neural network model of concept acquisition. *Cognitive Science*, 15(4):461–508, 1991.
- [81] Leslie B Cohen, Harold H Chaput, and Cara H Cashon. A constructivist model of infant cognition. *Cognitive Development*, 17(3-4):1323–1343, 2002.
- [82] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic apss and epsps. *Science*, 275(5297):213–215, 1997.
- [83] Li I Zhang, Huizhong W Tao, Christine E Holt, William A Harris, and Muming Poo. A critical window for cooperation and competition among developing retinotectal synapses. *Nature*, 395(6697):37–44, 1998.
- [84] Rudy Guyonneau, Rufin VanRullen, and Simon J Thorpe. Neurons tune to the earliest spikes through stdp. *Neural Computation*, 17(4):859–879, 2005.
- [85] Wulfram Gerstner, Richard Kempter, J Leo Van Hemmen, and Hermann Wagner. A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78, 1996.
- [86] Timothée Masquelier, Rudy Guyonneau, and Simon J Thorpe. Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains. *PloS one*, 3(1):e1377, 2008.

- [87] Timothée Masquelier, Rudy Guyonneau, and Simon J Thorpe. Competitive stdp-based spike pattern learning. *Neural computation*, 21(5):1259–1276, 2009.
- [88] Patrick D Roberts and Curtis C Bell. Computational consequences of temporally asymmetric learning rules: Ii. sensory image cancellation. *Journal of computational neuroscience*, 9(1):67–83, 2000.
- [89] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019.
- [90] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, and Timothée Masquelier. Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing*, 205:382–392, 2016.
- [91] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018.
- [92] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J Thorpe, and Timothée Masquelier. Combining stdp and reward-modulated stdp in deep convolutional spiking neural networks for digit recognition. *arXiv preprint arXiv:1804.00227*, 1, 2018.
- [93] Johannes C Thiele, Olivier Bichler, and Antoine Dupret. Event-based, timescale invariant unsupervised online deep learning with stdp. *Frontiers in computational neuroscience*, 12:46, 2018.
- [94] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, page 774, 2018.

- [95] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.
- [96] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [97] Jürgen Schmidhuber, Sepp Hochreiter, et al. Long short-term memory. *Neural Comput*, 9(8):1735–1780, 1997.
- [98] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [99] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.
- [100] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [101] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):1–10, 2016.
- [102] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.
- [103] Thomas Nowotny, James P Turner, and James C Knight. Loss shaping enhances exact gradient learning with eventprop in spiking neural networks. *arXiv preprint arXiv:2212.01232*, 2022.

- [104] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.
- [105] Sam McKeenoch, Dingding Liu, and Linda G Bushnell. Fast modifications of the spikeprop algorithm. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 3970–3977. IEEE, 2006.
- [106] Olaf Booij and Hieu tat Nguyen. A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, 95(6):552–558, 2005.
- [107] Samanwoy Ghosh-Dastidar and Hojjat Adeli. A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural networks*, 22(10):1419–1431, 2009.
- [108] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems*, 29(7):3227–3235, 2017.
- [109] Tao Liu, Zihao Liu, Fuhong Lin, Yier Jin, Gang Quan, and Wujie Wen. Mt-spike: A multilayer time-based spiking neuromorphic architecture with temporal error backpropagation. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 450–457. IEEE, 2017.
- [110] Robert Gütiğ and Haim Sompolinsky. The tempotron: a neuron that learns spike timing-based decisions. *Nature neuroscience*, 9(3):420–428, 2006.
- [111] Ran Rubin, Rémi Monasson, and Haim Sompolinsky. Theory of spike timing-based neural classifiers. *Physical review letters*, 105(21):218102, 2010.
- [112] Robert Gütiğ. Spiking neurons can discover predictive features by aggregate-label learning. *Science*, 351(6277):aab4113, 2016.

- [113] Jakub Fil and Dominique Chu. Minimal spiking neuron for solving multilabel classification tasks. *Neural Computation*, 32(7):1408–1429, 2020.
- [114] Răzvan V Florian. The chronotron: A neuron that learns to fire temporally precise spike patterns. *PLOS ONE*, 2012.
- [115] Filip Ponulak and Andrzej Kasiński. Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting. *Neural computation*, 22(2):467–510, 2010.
- [116] Ammar Mohemmed, Stefan Schliebs, Satoshi Matsuda, and Nikola Kasabov. Span: Spike pattern association neuron for learning spatio-temporal spike patterns. *International journal of neural systems*, 22(04):1250012, 2012.
- [117] Qiang Yu, Huajin Tang, Kay Chen Tan, and Haizhou Li. Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *Plos one*, 8(11):e78318, 2013.
- [118] Christian Albers, Maren Westkott, and Klaus Pawelzik. Learning of precise spike times with homeostatic membrane potential dependent synaptic plasticity. *PloS one*, 11(2):e0148948, 2016.
- [119] Brian Gardner and André Grüning. Supervised learning in spiking neural networks for precise temporal encoding. *PloS one*, 11(8):e0161335, 2016.
- [120] Ioana Sporea and André Grüning. Supervised learning in multilayer spiking neural networks. *Neural computation*, 25(2):473–509, 2013.
- [121] Malu Zhang, Hong Qu, Xiurui Xie, and Jürgen Kurths. Supervised learning in spiking neural networks with noise-threshold. *Neurocomputing*, 219:333–349, 2017.
- [122] Dheeru Dua and Casey Graff. Uci machine learning repository, 2017.

- [123] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE signal processing magazine*, 29(6):141–142, 2012.
- [124] Jean-Pascal Pfister, Taro Toyoizumi, David Barber, and Wulfram Gerstner. Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural computation*, 18(6):1318–1348, 2006.
- [125] Brian Gardner, Ioana Sporea, and André Grüning. Learning spatiotemporally encoded pattern transformations in structured spiking neural networks. *Neural computation*, 27(12):2548–2586, 2015.
- [126] Johanni Brea, Walter Senn, and Jean-Pascal Pfister. Matching recall and storage in sequence learning with spiking neural networks. *Journal of neuroscience*, 33(23):9565–9575, 2013.
- [127] Danilo Jimenez Rezende and Wulfram Gerstner. Stochastic variational learning in recurrent spiking networks. *Frontiers in Computational Neuroscience*, 8, 2014.
- [128] Hesham Mostafa and Gert Cauwenberghs. A learning framework for winner-take-all networks with stochastic synapses. *Neural computation*, 30(6):1542–1572, 2018.
- [129] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.
- [130] Priyadarshini Panda and Kaushik Roy. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 299–306. IEEE, 2016.

- [131] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.
- [132] Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.
- [133] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.
- [134] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.
- [135] Jianxiong Tang, Jianhuang Lai, Xiaohua Xie, Lingxiao Yang, and Wei-Shi Zheng. Snn2ann: A fast and memory-efficient training framework for spiking neural networks. *arXiv preprint arXiv:2206.09449*, 2022.
- [136] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. *Advances in neural information processing systems*, 28, 2015.
- [137] Simon Davidson and Steve B Furber. Comparison of artificial and spiking neural networks on digital hardware. *Frontiers in Neuroscience*, 15:651141, 2021.
- [138] Sander M Bohte. Error-backpropagation in networks of fractionally predictive spiking neurons. In *International Conference on Artificial Neural Networks*, pages 60–68. Springer, 2011.

- [139] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. From the cover: Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences of the United States of America*, 113(41):11441, 2016.
- [140] Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, 31, 2018.
- [141] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems*, 31, 2018.
- [142] Stanisław Woźniak, Angeliki Pantazi, Thomas Bohnstingl, and Evangelos Eleftheriou. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6):325–336, 2020.
- [143] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):1–15, 2020.
- [144] Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. Surrogate gradients for analog neuromorphic computing. *arXiv preprint arXiv:2006.07239*, 2020.
- [145] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021.

- [146] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the national academy of sciences*, 113(41):11441–11446, 2016.
- [147] Nicolas Perez-Nieves and Dan Goodman. Sparse spiking gradient descent. *Advances in Neural Information Processing Systems*, 34:11795–11808, 2021.
- [148] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.
- [149] Wolfgang Maass. Liquid state machines: motivation, theory, and applications. *Computability in context: computation and logic in the real world*, pages 275–296, 2011.
- [150] David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.
- [151] Brian DePasquale, Mark M Churchland, and LF Abbott. Using firing-rate dynamics to train recurrent networks of spiking model neurons. *arXiv preprint arXiv:1601.07620*, 2016.
- [152] Wilten Nicola and Claudia Clopath. Supervised learning in spiking neural networks with force training. *Nature communications*, 8(1):1–15, 2017.
- [153] Simon Haykin. *Neural networks and learning machines*, 3/E. Pearson Education India, 2009.
- [154] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

- [155] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- [156] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85, 2016.
- [157] Łukasz Kuśmierz, Takuya Isomura, and Taro Toyozumi. Learning with three factors: modulating hebbian plasticity with errors. *Current opinion in neurobiology*, 46:170–177, 2017.
- [158] Xiaohui Xie and H Sebastian Seung. Learning in neural networks by reinforcement of irregular spiking. *Physical Review E*, 69(4):041909, 2004.
- [159] Răzvan V Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural computation*, 19(6):1468–1502, 2007.
- [160] Eleni Vasilaki, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS computational biology*, 5(12):e1000586, 2009.
- [161] Wiebke Potjans, Abigail Morrison, and Markus Diesmann. A spiking neural network model of an actor-critic learning agent. *Neural computation*, 21(2):301–339, 2009.
- [162] Wiebke Potjans, Markus Diesmann, and Abigail Morrison. An imperfect dopaminergic error signal can drive temporal-difference learning. *PLoS computational biology*, 7(5):e1001133, 2011.

- [163] Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS computational biology*, 9(4):e1003024, 2013.
- [164] Eugene M Izhikevich. Solving the distal reward problem through linkage of stdp and dopamine signaling. *Cerebral cortex*, 17(10):2443–2452, 2007.
- [165] Zhenshan Bing, Claus Meschede, Kai Huang, Guang Chen, Florian Rohrbein, Mahmoud Akl, and Alois Knoll. End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4725–4732. IEEE, 2018.
- [166] Guangzhi Tang, Neelesh Kumar, Raymond Yoo, and Konstantinos Michmizos. Deep reinforcement learning with population-coded spiking neural network for continuous control. In *Conference on Robot Learning*, pages 2016–2029. PMLR, 2021.
- [167] Bleema Rosenfeld, Osvaldo Simeone, and Bipin Rajendran. Learning first-to-spike policies for neuromorphic control using policy gradients. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2019.
- [168] Devdhar Patel, Hananel Hazan, Daniel J Saunders, Hava T Siegelmann, and Robert Kozma. Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to atari breakout game. *Neural Networks*, 120:108–115, 2019.
- [169] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

- [170] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [171] Christof Koch and Gilles Laurent. Complexity and the nervous system. *Science*, 284(5411):96–98, 1999.
- [172] Julijana Gjorgjieva, Guillaume Drion, and Eve Marder. Computational implications of biophysical diversity and multiple timescales in neurons and synapses for circuit performance. *Current opinion in neurobiology*, 37:44–52, 2016.
- [173] Maoz Shamir and Haim Sompolinsky. Implications of neuronal diversity on population coding. *Neural computation*, 18(8):1951–1986, 2006.
- [174] Mircea I Chelaru and Valentin Dragoi. Efficient coding in heterogeneous neuronal populations. *Proceedings of the National Academy of Sciences*, 105(42):16344–16349, 2008.
- [175] Leslie C Osborne, Stephanie E Palmer, Stephen G Lisberger, and William Bialek. The neural basis for combinatorial coding in a cortical population response. *Journal of Neuroscience*, 28(50):13522–13531, 2008.
- [176] Gary Marsat and Leonard Maler. Neural heterogeneity and efficient population codes for communication signals. *Journal of neurophysiology*, 104(5):2543–2555, 2010.
- [177] Krishnan Padmanabhan and Nathaniel N Urban. Intrinsic biophysical diversity decorrelates neuronal firing while increasing information content. *Nature neuroscience*, 13(10):1276–1282, 2010.

- [178] Eric Hunsberger, Matthew Scott, and Chris Eliasmith. The competing benefits of noise and heterogeneity in neural coding. *Neural computation*, 26(8):1600–1623, 2014.
- [179] Fleur Zeldenrust, Boris Gutkin, and Sophie Denéve. Efficient and robust coding in heterogeneous recurrent networks. *PLoS computational biology*, 17(4):e1008673, 2021.
- [180] Johannes Lengler, Florian Jug, and Angelika Steger. Reliable neuronal systems: the importance of heterogeneity. *PloS one*, 8(12):e80694, 2013.
- [181] Zachary P Kilpatrick, Bard Ermentrout, and Brent Doiron. Optimizing working memory with heterogeneity of recurrent cortical excitation. *Journal of neuroscience*, 33(48):18999–19011, 2013.
- [182] Renato Duarte and Abigail Morrison. Leveraging heterogeneity for neural computation with fading memory in layer 2/3 cortical microcircuits. *PLoS computational biology*, 15(4):e1006781, 2019.
- [183] Guillaume Bellec, Franz Scherr, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv preprint arXiv:1901.09049*, 2019.
- [184] Haowen Fang, Amar Shrestha, Ziyi Zhao, and Qinru Qiu. Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network. *arXiv preprint arXiv:2003.02944*, 2020.
- [185] Ziv Rotman and Vitaly A Klyachko. Role of synaptic dynamics and heterogeneity in neuronal learning of temporal code. *Journal of neurophysiology*, 110(10):2275–2286, 2013.

- [186] Shengdun Wu, Yangsong Zhang, Yan Cui, Heng Li, Jiakang Wang, Lijun Guo, Yang Xia, Dezhong Yao, Peng Xu, and Daqing Guo. Heterogeneity of synaptic input connectivity regulates spike-based neuronal avalanches. *Neural Networks*, 110:91–103, 2019.
- [187] Dan FM Goodman, Victor Benichoux, and Romain Brette. Decoding neural responses to temporal cues for sound localization. *Elife*, 2:e01312, 2013.
- [188] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.
- [189] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [190] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.
- [191] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [192] Paul B Manis, Michael R Kasten, and Ruili Xie. Classification of neurons in the adult mouse cochlear nucleus: Linear discriminant analysis. *PloS one*, 14(10):e0223137, 2019.
- [193] Paul Manis, Michael R. Kasten, and Ruili Xie. Raw voltage and current traces for current-voltage (iv) relationships for cochlear nucleus neurons, 2019.

- [194] Ed S Lein, Michael J Hawrylycz, Nancy Ao, Mikael Ayres, Amy Bensinger, Amy Bernard, Andrew F Boe, Mark S Boguski, Kevin S Brockway, Emi J Byrnes, et al. Genome-wide atlas of gene expression in the adult mouse brain. *Nature*, 445(7124):168–176, 2007.
- [195] Michael J Hawrylycz, Ed S Lein, Angela L Guillozet-Bongaarts, Elaine H Shen, Lydia Ng, Jeremy A Miller, Louie N Van De Lagemaat, Kimberly A Smith, Amanda Ebbert, Zackery L Riley, et al. An anatomically comprehensive atlas of the adult human brain transcriptome. *Nature*, 489(7416):391–399, 2012.
- [196] Chao Huang, Andrey Resnik, Tansu Celikel, and Bernhard Englitz. Adaptive spike threshold enables robust and temporally precise neuronal encoding. *PLoS computational biology*, 12(6):e1004984, 2016.
- [197] Yulia Lerner, Christopher J Honey, Mikhail Katkov, and Uri Hasson. Temporal scaling of neural responses to compressed and dilated natural speech. *Journal of neurophysiology*, 111(12):2433–2444, 2014.
- [198] Florian Blättler and Richard HR Hahnloser. An efficient coding hypothesis links sparsity and selectivity of neural responses. *PloS one*, 6(10):e25506, 2011.
- [199] Shreejoy J Tripathy, Judith Savitskaya, Shawn D Burton, Nathaniel N Urban, and Richard C Gerkin. Neuroelectro: a window to the world’s neuron electrophysiology data. *Frontiers in neuroinformatics*, 8:40, 2014.
- [200] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [201] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

- [202] Sebastian Schmitt, Johann Klähn, Guillaume Bellec, Andreas Grübl, Maurice Guettler, Andreas Hartel, Stephan Hartmann, Dan Husmann, Kai Husmann, Sebastian Jeltsch, et al. Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 international joint conference on neural networks (IJCNN)*, pages 2227–2234. IEEE, 2017.
- [203] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2661–2671, 2021.
- [204] Silvan C Quax, Michele D’Asaro, and Marcel AJ van Gerven. Adaptive time scales in recurrent neural networks. *Scientific reports*, 10(1):1–14, 2020.
- [205] Bojian Yin, Federico Corradi, and Sander M Bohté. Effective and efficient computation with multiple-timescale spiking recurrent neural networks. In *International Conference on Neuromorphic Systems 2020*, pages 1–8, 2020.
- [206] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.
- [207] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai. *Commun. ACM*, 63(12):54–63, November 2020.
- [208] Jacob Hochstetler, Rahul Padidela, Qi Chen, Qing Yang, and Song Fu. Embedded deep learning for vehicular edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 341–343. IEEE, 2018.
- [209] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing*, 16(3):82–88, 2017.

- [210] David Daniel Cox and Thomas Dean. Neural networks and neuroscience-inspired computer vision. *Current Biology*, 24(18):R921–R929, 2014.
- [211] Thomas Pfeil, Andreas Grübl, Sebastian Jeltsch, Eric Müller, Paul Müller, Mihai A Petrovici, Michael Schmuker, Daniel Brüderle, Johannes Schemmel, and Karlheinz Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in neuroscience*, 7:11, 2013.
- [212] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.
- [213] Misha Mahowald and Rodney Douglas. A silicon neuron. *Nature*, 354(6354):515–518, 1991.
- [214] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950. IEEE, 2010.
- [215] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *2011 IEEE custom integrated circuits conference (CICC)*, pages 1–4. IEEE, 2011.
- [216] Saber Moradi and Giacomo Indiveri. An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE transactions on biomedical circuits and systems*, 8(1):98–107, 2013.

- [217] Elisabetta Chicca, Fabio Stefanini, Chiara Bartolozzi, and Giacomo Indiveri. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE*, 102(9):1367–1388, 2014.
- [218] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.
- [219] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.
- [220] G Pedretti, V Milo, S Ambrogio, R Carboni, S Bianchi, A Calderoni, N Ramaswamy, AS Spinelli, and D Ielmini. Memristive neural network for on-line learning and tracking with brain-inspired spike timing dependent plasticity. *Scientific reports*, 7(1):1–10, 2017.
- [221] Chul-Heung Kim, Soochang Lee, Sung Yun Woo, Won-Mook Kang, Suhwan Lim, Jong-Ho Bae, Jaeha Kim, and Jong-Ho Lee. Demonstration of unsupervised learning with spike-timing-dependent plasticity using a tft-type nor flash memory array. *IEEE Transactions on Electron Devices*, 65(5):1774–1780, 2018.
- [222] Dongseok Kwon, Suhwan Lim, Jong-Ho Bae, Sung-Tae Lee, Hyeongsu Kim, Young-Tak Seo, Seongbin Oh, Jangsaeng Kim, Kyuho Yeom, Byung-Gook Park, et al. On-chip training spiking neural networks using approximated back-propagation with analog synaptic devices. *Frontiers in neuroscience*, 14:423, 2020.

- [223] Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yan-nik Stradmann, Johannes Weis, et al. Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate. *arXiv preprint arXiv:2006.07239*, 2020.
- [224] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.
- [225] Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Neural heterogeneity promotes robust learning. *Nature communications*, 12(1):1–9, 2021.
- [226] Lei Jimmy Ba Brendan Frey. Adaptive dropout for training deep neural networks. *Proceeding NIPS*, 13:3084–3092, 2013.
- [227] Guy Blanc and Steffen Rendle. Adaptive sampled softmax with kernel based sampling. In *International Conference on Machine Learning*, pages 590–599. PMLR, 2018.
- [228] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *arXiv preprint arXiv:1903.03129*, 2019.
- [229] Shabnam Daghighi, Nicholas Meisburger, Mengnan Zhao, and Anshumali Shrivastava. Accelerating slide deep learning on modern cpus: Vectorization, quantizations, memory optimizations, and more. *Proceedings of Machine Learning and Systems*, 3, 2021.
- [230] Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: exper-

- imental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018.
- [231] James C Knight and Thomas Nowotny. Gpus outperform current hpc and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Frontiers in neuroscience*, 12:941, 2018.
- [232] Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):1379–1397, 2015.
- [233] Yigit Demirag, Filippo Moro, Thomas Dalgaty, Gabriele Navarro, Charlotte Frenkel, Giacomo Indiveri, Elisa Vianello, and Melika Payvand. Pcm-trace: Scalable synaptic eligibility traceswith resistivity drift of phase-change materials. *arXiv preprint arXiv:2102.07260*, 2021.
- [234] Yanqi Chen, Zhaofer Yu, Wei Fang, Tiejun Huang, and Yonghong Tian. Pruning of deep spiking neural networks through gradient rewiring. *arXiv preprint arXiv:2105.04916*, 2021.
- [235] Isabella Pozzi, Sander Bohte, and Pieter Roelfsema. Attention-gated brain propagation: How the brain can implement reward-based error backpropagation. *Advances in Neural Information Processing Systems*, 33, 2020.
- [236] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [237] Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, et al. A deep learning framework for neuroscience. *Nature neuroscience*, 22(11):1761–1770, 2019.
- [238] Friedemann Zenke, Sander M Bohtë, Claudia Clopath, Iulia M Comşa, Julian Göltz, Wolfgang Maass, Timothée Masquelier, Richard Naud, Emre O

- Neftci, Mihai A Petrovici, et al. Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron*, 109(4):571–575, 2021.
- [239] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [240] Julian Rossbroich, Julia Gyga, and Friedemann Zenke. Fluctuation-driven initialization for spiking neural network training. *Neuromorphic Computing and Engineering*, 2(4):044016, 2022.
- [241] Maciej Skorski, Alessandro Temperoni, and Martin Theobald. Revisiting weight initialization of deep neural networks. In *Asian Conference on Machine Learning*, pages 1192–1207. PMLR, 2021.
- [242] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [243] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [244] Bing Xu, Ruitong Huang, and Mu Li. Revise saturated activation functions. *arXiv preprint arXiv:1602.05980*, 2016.
- [245] Devansh Arpit, Víctor Campos, and Yoshua Bengio. How to initialize your network? robust initialization for weightnorm & resnets. *Advances in Neural Information Processing Systems*, 32, 2019.
- [246] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How

- to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018.
- [247] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.
- [248] Magnus JE Richardson and Rupert Swarbrick. Firing-rate response of a neuron receiving excitatory and inhibitory synaptic shot noise. *Physical review letters*, 105(17):178102, 2010.
- [249] Magnus JE Richardson. Firing-rate response of linear and nonlinear integrate-and-fire neurons to modulated current-based and conductance-based synaptic drive. *Physical Review E*, 76(2):021919, 2007.
- [250] Magnus JE Richardson. Spike-train spectra and network response functions for non-linear integrate-and-fire neurons. *Biological Cybernetics*, 99(4):381–392, 2008.
- [251] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *eLife*, 8:e47314, August 2019.
- [252] Oliver Ibe. *Markov processes for stochastic modeling*. Newnes, 2013.
- [253] Kyle Siegrist. *Probability, Mathematical Statistics, and Stochastic Processes*. LibreTexts Statistics, 2021. Online. Accessed December 18, 2022 at [https://stats.libretexts.org/Bookshelves/Probability_Theory/Probability_Mathematical_Statistics_and_Stochastic_Processes_\(Siegrist\)](https://stats.libretexts.org/Bookshelves/Probability_Theory/Probability_Mathematical_Statistics_and_Stochastic_Processes_(Siegrist)).
- [254] Monroe David Donsker. An invariance principle for certain probability limit theorems. *Memoirs of the American Mathematical Society*, 6, 1951.
- [255] Steven Lalley. Lecture notes in mathematical finance. lecture 5, 2001. <http://galton.uchicago.edu/~lalley/Courses/390/Lecture5.pdf>.

- [256] Steve Lalley and Per Mykland. Lecture notes in stochastic processes ii. lecture g: Brownian motion, 2013. <https://galton.uchicago.edu/~lalley/Courses/313/BrownianMotionCurrent.pdf>.
- [257] David Zwicker. py-pde: A python package for solving partial differential equations. *Journal of Open Source Software*, 5(48):2158, 2020.
- [258] Bojian Yin, Federico Corradi, and Sander M Bohtë. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nature Machine Intelligence*, 3(10):905–913, 2021.
- [259] Chloe Winston, Dana Mastrovito, Eric Shea-Brown, and Stefan Mihalas. Heterogeneity in neuronal dynamics is learned by gradient descent for temporal processing tasks. *bioRxiv*, 2022.
- [260] Marcello Salustri, Shunra Yoshida, and Ruggero Micheletto. Neural and axonal heterogeneity improves information transmission. *Available at SSRN 4266707*, 2022.
- [261] Jesse Hagenaars, Federico Paredes-Vallés, and Guido De Croon. Self-supervised learning of event-based optical flow with spiking neural networks. *Advances in Neural Information Processing Systems*, 34:7167–7179, 2021.
- [262] Nada Y Abdelrahman, Eleni Vasilaki, and Andrew C Lin. Compensatory variability in network parameters enhances memory performance in the drosophila mushroom body. *Proceedings of the National Academy of Sciences*, 118(49):e2102158118, 2021.
- [263] Aitor Morales-Gregorio, Alexander van Meegen, and Sacha Jennifer van Albeda. Ubiquitous lognormal distribution of neuron densities across mammalian cerebral cortex. *bioRxiv*, 2022.

- [264] Briar Doty, Stefan Mihalas, Anton Arkhipov, and Alex Piet. Heterogeneous ‘cell types’ can improve performance of deep neural networks. *bioRxiv*, 2021.
- [265] Anshul Choudhary, Anil Radhakrishnan, John F Lindner, Sudeshna Sinha, and William L Ditto. Neural networks embrace learned diversity. *arXiv preprint arXiv:2204.04348*, 2022.
- [266] Chao Jin, Chen Wang, Xin Qi, Zhijun Wang, Yuan He, and Lianchun Yu. Criticality and heterogeneity as general principles to optimize performance for spiking cameras. *ResearchSquare preprint*, 2022.
- [267] Luke Taylor, Andrew King, and Nicol Harper. Accelerating spiking neural network training. *arXiv preprint arXiv:2205.15286*, 2022.
- [268] Anand Subramoney, Khaleelulla Khan Nazeer, Mark Schöne, Christian Mayr, and David Kappel. Egru: Event-based gru for activity-sparse inference and learning. *arXiv preprint arXiv:2206.06178*, 2022.
- [269] Yaoyu Zhu, Zhaofei Yu, Wei Fang, Xiaodong Xie, Tiejun Huang, and Timothée Masquelier. Training spiking neural networks with event-driven backpropagation. In *Advances in Neural Information Processing Systems*, 2022.
- [270] Sheldon M Ross, John J Kelly, Roger J Sullivan, William James Perry, Donald Mercer, Ruth M Davis, Thomas Dell Washburn, Earl V Sager, Joseph B Boyce, and Vincent L Bristow. *Stochastic processes*, volume 2. Wiley New York, 1996.