Thèse n° 10 637

Supervised learning and inference of spiking neural networks with temporal coding

Présentée le 13 décembre 2023

Faculté informatique et communications
Laboratoire de calcul neuromimétique (IC/SV)
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Ana STANOJEVIC

Acceptée sur proposition du jury

Prof. P. Thiran, président du jury
Prof. W. Gerstner, Dr S. A. Wozniak, directeurs de thèse
Prof. S.-C. Liu, rapporteuse
Prof. F. Zenke, rapporteur
Prof. N. Flammarion, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

"There is nothing permanent except change."
— Heraclitus

To all people who believed in me

Acknowledgements

My first thanks goes to my advisor Prof. Wulfram Gerstner, without whom this thesis wouldn't have been possible. During my master's studies at EPFL he was the one who inspired me to pursue a PhD by exemplifying how exciting research is and by embodying the true scientist. I am deeply grateful for the opportunity to work with Wulfram and learn from him. He has truly exceeded my expectations as a phantastic researcher but even more importantly as a reliable and supportive mentor. I would also like to express gratitude to the jury president Prof. Patrick Thiran and the jury experts, Prof. Shih-Chi Liu, Prof. Friedemann Zenke and Prof. Nicolas Flammarion, for their interest in the thesis.

My second thanks goes to my former co-advisor Prof. Evangelos Eleftheriou, for providing me the opportunity to be a part of the IBM Research community. Evangelos is a visionary who understands research and the scientific community on a much deeper level, making it a tremendous pleasure to talk to him, work with him and learn from him. I am particularly grateful for his support and genuine interest in my research and progress. I also thank Dr. Giovanni Cherubini, Dr. Abu Sebastian and Dr. Timoleon Moraitis who initially worked with me as a master's student in IBM Research, setting a pathway for successful collaboration.

I would like to offer my appreciation to Dr. Guillaume Bellec, for selflessly bringing his ideas and expertise to my project. His way of thinking and presenting has taught me a lot and significantly increased the quality of the thesis. My gratitude also goes to my current co-advisor Dr. Stanisław Woźniak, who has supported me by consistently offering feedback on ideas and results. Finally, I am thankful to my manager Dr. Angeliki Pantazi for her responsiveness and for ensuring that complex and demanding processes are promptly resolved.

It is essential to acknowledge the IBM Research family, who made every day of this PhD journey truly enjoyable. A warm thanks to my friends Dr. Geethan Karunaratne, Jesse Luchtenveld, Jovin Langenegger, Michael Hersche, Dr. Abbas Rahimi, Dr. Ghazi Sarwat Syed, Oleh Kolner, Abhairaj Singh, Dr. Benedikt Kersting and Francesco Carzaniga for sharing countless lunch breaks and even more laughs. I am deeply grateful for my dear Dr. Irem Boybat, Dr. Mridula Prathapan and Dr. Giorgia Dellaferrera who were a supportive female squad during times when I needed it the most. A big gratitude also to Dr. Burkhard Ringlein for sharing many coffee breaks and engaging in the most interesting discussions. A very special thanks goes to my dear friend Dr. Jovan Blanuša, with whom every conversation was both warm and incredibly helpful. Finally, I am grateful to all the members of the LCN group for being welcoming, knowledgeable and open for discussions.

A profound feeling of appreciation overwhelms me for all my friends in Zürich who made me feel at home. My love goes to Ivana Radić, Deborah Aeberhard, Lina Bardaje, Alienor Auge, Chenqi Jia, Lisa Jenny, Nina Desnica, Antonija Burčul and Petar Šegina. They have transformed moments in my life into cherished memories. Special thanks to Deborah Aeberhard who inspired me to learn German, to Alienor Auge for always inquiring about the title of my PhD thesis and to Michael Künzli for translating the abstract.

Finally, I would like to express my gratitude to my mother Snežana for her emotional support and for being an anchor and to my late father Goran for instilling in me the value of education and hard work. An immense appreciation belongs to my primary school teachers, who made science an exciting and engaging experience, unknowingly setting me on a pathway to pursue a PhD. Warm thanks also to my sister Marija for pushing me out of my comfort zone my whole life and to my lifelong friend Sava Stevanović. Last but certainly not least, I am forever grateful to my grandparents who gave me all their love and made me the person I am today.

Lausanne, August 2, 2023 A. S.

Abstract

The way biological brains carry out advanced yet extremely energy efficient signal processing remains both fascinating and unintelligible. It is known however that at least some areas of the brain perform fast and low-cost processing relying only on a small number of temporally encoded spikes. This thesis investigates supervised learning and inference of spiking neural networks (SNNs) with sparse temporally encoded communication. We explore different setups and compare the performance of our SNNs with that of the standard artificial neural networks (ANNs) on data classification tasks.

In the first setup we consider: *A family of exact mappings between a single-spike network and a ReLU network*. We dismiss training for a moment and analyse deep SNNs with time-to-first-spike (TTFS) encoding. There exist a neural dynamics and a set of parameter constraints which guarantee an approximation-free mapping (conversion) from a ReLU network to an SNN with TTFS encoding. We find that a pretrained deep ReLU network can be replaced with our deep SNN without any performance loss on large-scale image classification tasks (CIFAR100 and PLACES365). However, we hypothesise that in many cases there is a need for training or fine-tuning deep spiking neural network for the specific problem at hand.

In the second setup we consider: *Training a deep single-spike network using a family of exact mappings from a ReLU network*. We thoroughly investigate the reasons for unsuccessful training of deep SNNs with TTFS encoding and uncover an instance of the vanishing-and-exploding gradient problem. We find that a particular exact mapping solves this problem and yields an SNN with learning trajectories equivalent to those of ReLU network on large image classification tasks (CIFAR100 and PLACES365). Training is crucial for fine-tuning SNNs for the specific device properties such as low latency, the amount of noise or quantization. We hope that this study will eventually lead to an SNN hardware implementation offering a low-power inference with ANN performance on data classification tasks.

Keywords spiking neural network, temporal encoding, sparse communication, efficient data classification, multiplication-free inference, backpropagation training, time-to-first-spike encoding, deep ReLU network conversion, SNN-ReLU network equivalence, deep SNN training

Zusammenfassung

Die Art und Weise, wie biologische Gehirne fortgeschrittene und zugleich äußerst energieeffiziente Signalverarbeitung durchführen, bleibt sowohl faszinierend als auch unverständlich. Es ist jedoch bekannt, dass zumindest einige Bereiche des Gehirns schnelle und kostengünstige Verarbeitungsprozesse durchführen, die sich nur auf eine kleine Anzahl zeitlich codierter Spitzen (spikes) stützt. Diese Arbeit untersucht das überwachte Lernen und die Inferenz von gepulsten neuronalen Netzwerken (Spiking Neural Networks, fortan SNNs) mit sparsamer zeitlich codierter Kommunikation. Wir erkunden verschiedene Konfigurationen und vergleichen die Leistung unserer SNNs mit der von herkömmlichen künstlichen neuronalen Netzwerken (Artificial Neural Networks, fortan ANNs) bei der Klassifizierung von Daten.

In der ersten Konfiguration betrachten wir: *Eine Familie exakter Zuordnungen zwischen einem Einzel-Spike-Netzwerk und einem ReLU-Netzwerk*. Wir verzichten vorerst auf das Training und analysieren tiefe SNNs mit Time-to-First-Spike"(TTFS) Codierung. Es existieren eine neuronale Dynamik und eine Reihe von Parameterbeschränkungen, die eine approximationsfreie Zuordnung (Konversion) von einem ReLU-Netzwerk zu einem SNN mit TTFS-Codierung garantieren. Wir stellen fest, dass ein vortrainiertes tiefes ReLU-Netzwerk durch unser tiefes SNN ohne Leistungsverlust bei Aufgaben zur Bildklassifizierung (CIFAR100 und PLACES365) ersetzt werden kann. Dennoch vermuten wir, dass es in vielen Fällen notwendig ist, ein tiefes Spiking Neural Network für das spezifische Problem zu trainieren oder anzupassen.

In der zweiten Konfiguration betrachten wir: *Das Training eines tiefen Einzel-Spike-Netzwerks mithilfe einer Familie von exakten Zuordnungen von einem ReLU-Netzwerk*. Wir untersuchen die Gründe für das erfolglose Training von tiefen SNNs mit TTFS-Codierung sorgfältig und decken ein Beispiel des Problems des Verschwindens und Explodierens des Gradienten auf. Wir stellen fest, dass eine bestimmte exakte Zuordnung dieses Problems löst und ein SNN mit Lerntrajektorien erzeugt, die denen eines ReLU-Netzwerks bei Aufgaben zur Bildklassifizierung (CIFAR100 und PLACES365) äquivalent sind. Die beobachtete Lernfähigkeit ist entscheidend für das Feinabstimmen von SNNs für spezifische Geräteeigenschaften wie geringe Latenzzeit, die Menge an Rauschen oder Quantisierung. Wir hoffen, dass diese Arbeit eine SNN-Hardwareimplementierung fördert, die eine energiesparende Inferenz mit ANN-Leistung bei der Klassifizierung von Daten ermöglicht.

Stichworte gepulstes neuronales Netzwerk, zeitliche Codierung, sparsame Kommunikation, effiziente Datenklassifizierung, multiplikationsfreie Inferenz, Backpropagation-Training, Time-to-First-Spike-Codierung, Konversion von tiefen ReLU-Netzwerken, Äquivalenz von SNN und ReLU-Netzwerk, Training von tiefen SNNs.

Contents

Acknowledgements **i**

Abstract (English/Deutsch) **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Spike encoding 2

1.3 Spiking neural networks (SNNs) 4

1.4 SNNs with temporal coding 8

1.5 Image classification task 10

1.6 Thesis structure 12

1.7 Author contributions 13

2 An exact mapping from ReLU networks to spiking neural networks **15**

2.1 Introduction 17

2.2 Results 19

2.2.1 Main Theoretical Result 19

2.2.2 Proof Sketch of Main Theoretical Result 21

2.2.3 Examples of equivalent mappings 25

2.2.4 Performance on Benchmark Datasets 27

2.3 Discussion 33

2.4 Methods 35

2.4.1 Preprocessing 35

2.4.2 Conversion to SNN 39

2.4.3 Conversion of Max pooling 42

2.4.4 Mapping of the output layer 42

2.4.5 Final remarks regarding the mapping 43

2.4.6 Datasets 43

3 Are training trajectories of deep single-spike and deep ReLU network equivalent? **47**

3.1 Introduction 48

3.2 Definition and properties of time-to-first-spike networks 50

3.3 Analysis of learning dynamics 54

3.4 Benchmark results 57

3.5 Discussion and future work 59
3.6 Supplemental information 61
  3.6.1 Generalization to other neuronal dynamics 61
  3.6.2 Setting $t_{\text{max}}^{(n)}$ and the threshold 62
  3.6.3 Simulation details 64

4 Conclusion **69**

A Additional publications **71**
A.1 Approximating ReLU networks by single-spike computation 71
  A.1.1 Introduction 73
  A.1.2 Method 75
  A.1.3 Results 77
  A.1.4 Conclusion 79
A.2 Time-encoded multiplication-free spiking neural networks: application to data classification tasks 80
  A.2.1 Introduction 82
  A.2.2 System Architecture 84
  A.2.3 Time Encoding of Input and Target Patterns 90
  A.2.4 Simulation Results 97
  A.2.5 Conclusions and future work 108
A.3 Design of Time-Encoded Spiking Neural Networks in 7nm CMOS Technology . 110

Bibliography **124**

Curriculum Vitae **125**

1 Introduction

1.1 Motivation

State-of-the-art artificial intelligence (AI) models are now being utilized for applications that were difficult to imagine only a decade ago. Some of the biggest accomplishments include self-driving cars [Feng et al. (2023)], robotic control [Hwangbo et al. (2019)], medical diagnostics [Yala et al. (2019)], being a world champion in Go [Silver et al. (2016)], generating images from a text description [Ramesh et al. (2021)] and of course writing human-like text to lead discussions on almost any topic [Brown et al. (2020)]. Nonetheless, there are still many challenges that standard artificial neural network (ANN) models experience, which are straightforward for humans. For example, the human brain is able to learn multiple tasks sequentially without catastrophic forgetting [Parisi et al. (2019)]. Moreover, generalization from a few data samples is relatively uncomplicated [Barnett and Ceci (2002)], and correct decisions are made even when there are errors in the input [Geirhos et al. (2018)].

Most state-of-the-art ANNs which have impressive performance on the previously mentioned tasks depend on a large amount of data and high computational power for their training and inference [Brown et al. (2020)]. Therefore, these models are 'locked in' on large, power-hungry servers and are not portable in their full capacity to edge devices with small batteries. [Jiang et al. (2018); Wang et al. (2020)]. As processing data near the location where it was collected offers significant benefits, there has been a long-standing effort in machine learning to reduce the size of ANN models to meet these constraints. Particularly interesting is a line of research investigating binarized neural networks (BNNs), where neuron outputs and weights take values of $+1$ or -1 [Courbariaux et al. (2016)]. While this approach reduces computational energy, it often results in a significant loss of performance.

The human brain is a network of around 86 billion neurons which communicate via sparse action potentials, which we may consider as binary spikes. Despite being an extremely advanced signal processing system, this biological neural network requires only 20W for its functioning (see Fig. 1.1). The standard ANN architectures and computing principles were inspired by the brain in the past [McCulloch and Pitts (1943); Rosenblatt (1958)]. Therefore,

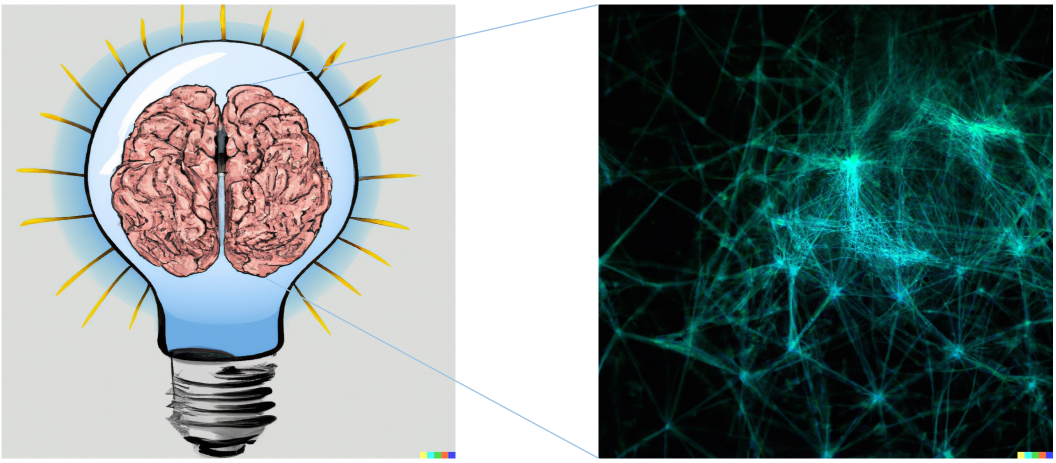


Figure 1.1: The brain is a network of neurons that operates on the power equivalent to that of an average light bulb. Created by DALL·E.

there is a good reason to believe that by modeling neural networks with spike communication and powerful brain-inspired dynamics, we could achieve energy-efficient processing without compromising the performance of modern ANNs. In such a scenario, AI would truly become ubiquitous.

We will now describe the main concepts observed in the brain, which are leveraged to create models exhibiting low-power advanced processing. The introduction aims to provide a general overview of the topic, while detailed explanations can be found in the subsequent chapters and by accessing the provided references. In Section 1.2, we discuss potential spike encoding mechanisms in the brain, emphasizing the efficient sparse temporal encoding scheme. In Section 1.3, we describe spiking neural network (SNN) models and various learning algorithms. Notably, Section 1.4 provides background information on SNNs with temporal coding, while Section 1.5 offers a brief introduction to the image classification task and the datasets we utilize. Finally, Sections 1.6 and 1.7 outline the thesis structure and detail the contributions made by the author.

1.2 Spike encoding

What is spike encoding? Even though we are aware that in the signal processing pipeline some cells might send analog signals to their neighbours [Kolb et al. (1995); Alle and Geiger (2006)], the predominant mode of communication in the brain is binary. Information is transmitted down the axon of the source neuron and through synapses to the input dendrites of multiple target neurons. In most cases these connections remain silent (equivalent to a zero value), and only sporadically do they transmit a short action potential which is always of the same magnitude (equivalent to a value of one), see Fig. 1.2a. A more detailed explanation of neuronal dynamics and the generation of action potentials is provided in Section 1.3. There are several possibilities why this particular mode of communication was prioritised by evolution,

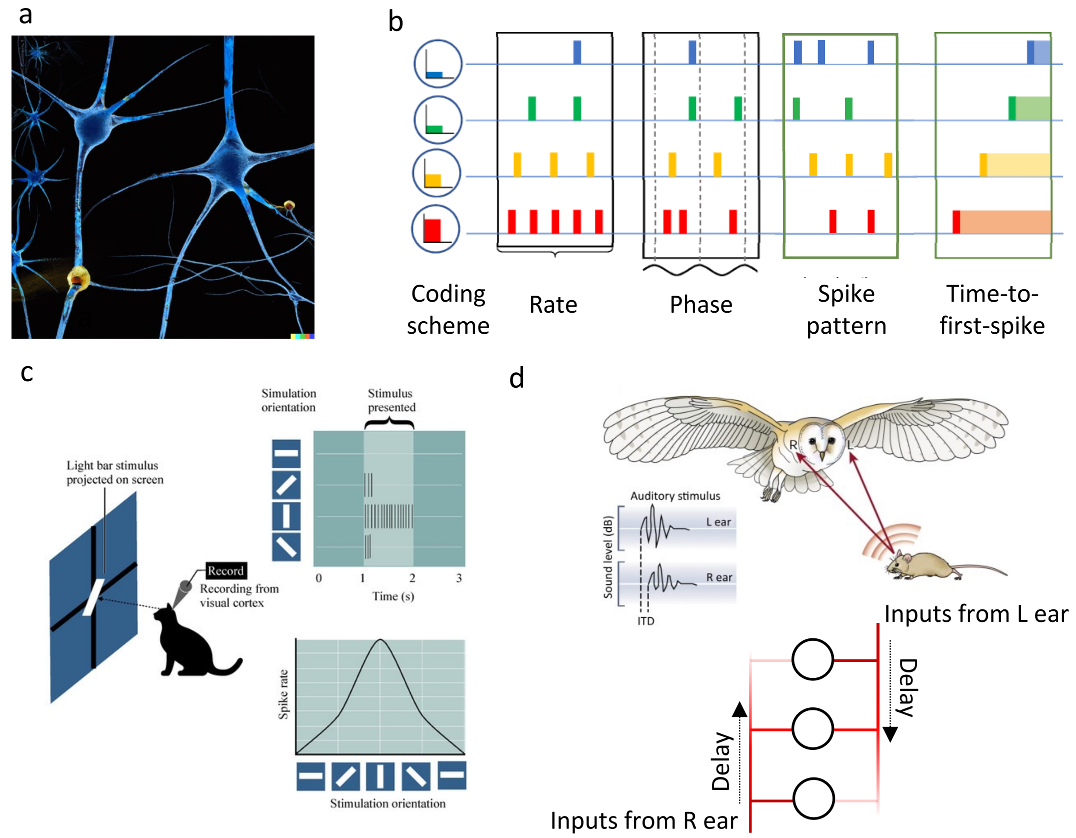


Figure 1.2: **Spike encoding. a.** Neurons in the brain communicate through sparse spikes (yellow). The question arises: what information does each spike represent? Created by DALL·E. **b.** An illustration of how popular spike encoding schemes represent inputs of varying intensities. Adapted from [Park et al. (2020)]. **c.** Rate coding detection in the brain of a cat, reproduced from [Li et al. (2022)]. **d.** Detection of source location in the horizontal plane by temporal coding in the brain of a barn owl, part of image reproduced from [Grothe (2018)].

most notably: (i) the exceptional energy efficiency it offers and (ii) its robustness in long-distance signal transmission. Indeed, the transmission of all-or-none signals enables low-cost communication that is more resilient to noise.

When a neuron receives a spike train over time (a sequence of zeros and ones), a question arises: what kind of information can be decoded or understood from such input? In Fig. 1.2b we observe how the same analog value can be interpreted from different sets of spikes by following various proposed coding schemes. Each scheme adheres to specific rules for encoding different analog values (color-coded). However, there is no consensus among scientists regarding how information is represented in the brain. In fact, it has been observed that different areas of the brain rely on different coding schemes, and the same spike train can carry the meaning of multiple encoding mechanisms [Gollisch and Meister (2008)].

Rate coding In the late fifties, experiments were conducted to record the outputs of specific neurons in the cat's brain while presenting certain visual stimuli, see Fig. 1.2c [Hubel and Wiesel (1959)]. When images of a light bar were displayed on the screen, scientists observed that depending on the orientation of the bar, some neurons generated a smaller or a larger number of spikes. This led to the conclusion that the information about bar orientation is encoded in the spiking rate of these neurons. Note that the typical spiking frequency in the brain ranges from 1Hz to 50Hz. Fig. 1.2b illustrates that with the rate coding scheme, larger analogue values are represented by a higher number of spikes.

Temporal coding A neuron requires a significant amount of time to estimate the spiking rate of its input. However, experiments conducted in the mid-nineties suggested that the classification of complex natural images occurs very rapidly, within 150ms [Thorpe et al. (1996)]. Given the multiple layers of processing and the typical spiking frequency, rate coding alone cannot achieve the fast classification observed in these experiments. As a result, not only the spiking rate but also the precise timing of spikes must carry the relevant information. This gave rise to a new class of viable and efficient coding schemes known as temporal coding.

Temporally coded information was discovered in various brain areas dedicated to sensory information processing [Kubke et al. (2002); Johansson and Birznieks (2004); O'Keefe and Recce (1993)]. For instance, experiments with barn owls (see Fig. 1.2d) demonstrate that sound localisation is determined by the interaural time difference (ITD). In other words, the position of the sound source can be inferred based on the timing of the signal reaching the neurons. Fig. 1.2b depicts three examples of temporal coding schemes: (i) phase coding, where the information is contained in the timing relative to a periodic signal; (ii) spatio-temporal spike pattern coding, where the information is contained in the timing relative to spikes across different neurons [Abeles (1982)]; and (iii) time-to-first-spike coding, where the information is contained in timing relative to some other time reference, such as stimulus onset.

Our work focuses on two schemes. Chapters 2, 3 and Section A.1 rely on time-to-first-spike (TTFS) coding, whereas Section A.2 assumes spatio-temporal spike pattern coding.

1.3 Spiking neural networks (SNNs)

There is evidence that information processing in the brain occurs in a hierarchical manner [Yamins and DiCarlo (2016)]. For example, the processing of an image typically begins in area V1, continues to V2 and V4 and then progresses to MT and IT areas, see Fig. 1.3a. Along this pathway, each neuron receives action potentials (spikes) from the previous layer through its dendrites, see Fig. 1.3b. It also receives many inputs from neurons in the same brain area and feedback connections from hierarchically higher areas, but we do not consider these. The membrane potential of the neuron changes based on the input it receives from other neurons and the strength of the corresponding synapses. When the membrane potential depolarizes sufficiently, an action potential is generated and transmitted through the neuron's axon. This results in a spike train that is sent to the subsequent neurons in the network.

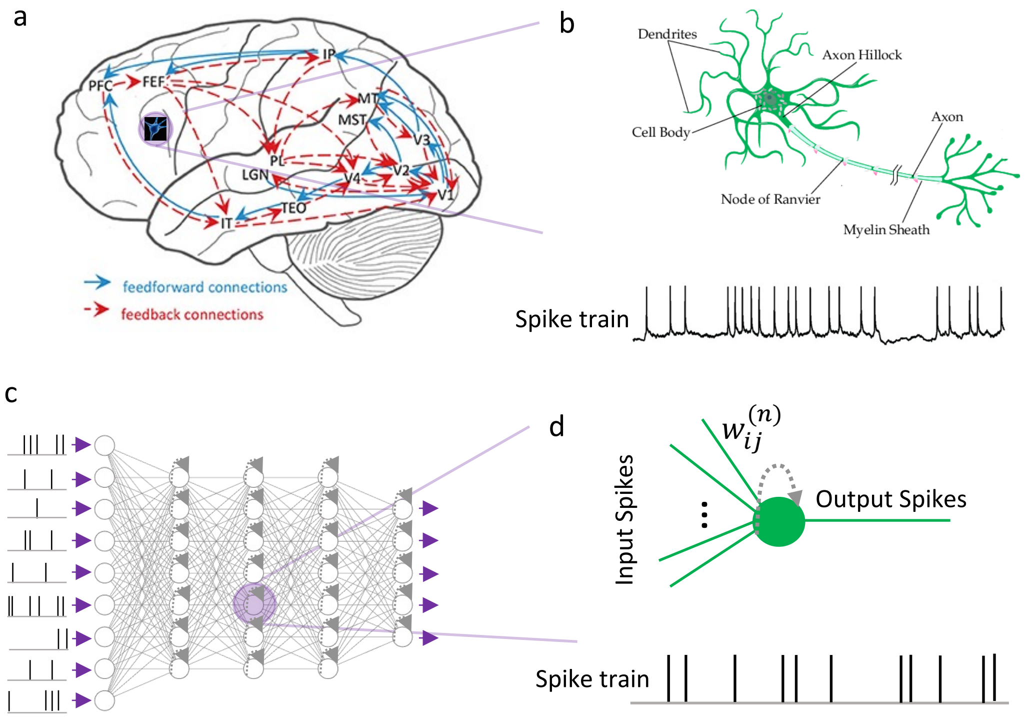


Figure 1.3: **Spiking neural network. a.** After the input stimuli are received, their processing occurs hierarchically across multiple brain areas (blue). Reproduced from [Zhao et al. (2020)]. **b.** Within each brain area, neurons receive input spikes through their dendrites and generate a spike train which is transmitted down the axon. Reproduced from [Ghahari and Enderle (2014)] **c.** An architecture of a spiking neural network (SNN) captures this hierarchical processing. **d.** Within the SNN, each neuron receives input spikes from the previous layer through synaptic connections with strengths denoted as $w_{ij}^{(n)}$. The neuron then generates its own spike train at the output.

To model hierarchical processing in the brain, we adopt a layered architecture similar to the one used by the standard artificial neural networks (ANNs), see Fig. 1.3c. The input layer generates spike trains which are transmitted to the first hidden layer, and so on. The neuronal dynamics is modeled using the (leaky) integrate-and-fire model [Gerstner et al. (2014)]. Upon receiving spikes, the neurons integrate them into their state variable (potential). The synaptic strength (weight) $w_{ij}^{(n)}$ determines the impact of spikes coming from neuron j in layer $n-1$ on the potential variable of neuron i in layer n, see Fig. 1.3d. Whenever the potential reaches a certain threshold value $\vartheta_i^{(n)}$, the neuron fires and the spike is transmitted to the neurons in the subsequent layer. After spiking, the potential of the neuron is reset to its resting value, typically 0, unless specified otherwise. In Chapters 2, 3 and Section A.1, we also incorporate a refractory period, which introduces a specific time interval during which the neuron cannot spike again. Moreover, in Chapters 2 and 3 the neurons in the output layer do not fire, but the value of their potential is read out.

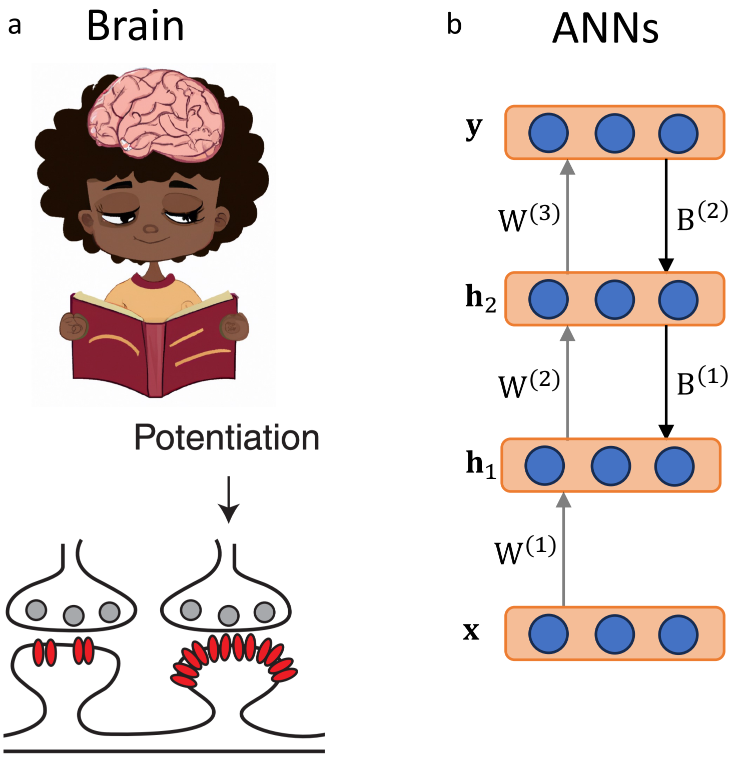


Figure 1.4: **Learning in the brain and standard ANNs. a.** When a child reads a book, certain synapses in the brain can strengthen by increasing the number of communication channels (bottom). The illustration at the top is created by DALL·E., whereas the one at the bottom is reproduced from [Turrigiano (2012)]. **b.** Training of a standard feed-forward ANN where the error is propagated backwards using the matrix $B^{(n)}$. In the case of backpropagation, $B^{(n)}$ corresponds to the transposed matrix $W^{(n)}$.

Plasticity in the brain It is well-known that young children learn rapidly due to the high plasticity of their brains. The brain is capable of changing itself by strengthening (creating) or weakening (removing) synapses based on the processing of input information and experiences, see Fig. 1.4a. The study of learning in the brain originated with the influential Hebbian rule, which states that synapses between neurons that are active together become stronger [Hebb (1949)]. Subsequent research has revealed that not only the coincidence but also the timing of activity between two neurons is crucial. Specifically, if the presynaptic (source) neuron fires after the postsynaptic (target) neuron, their connection will be weakened; reviewed in [Bi and Poo (2001)] and [Markram et al. (2011)]. Over the years, various local learning rules have been proposed to model Hebbian learning and STDP [Oja (1982); Bienenstock et al. (1982); Pfister and Gerstner (2006); Graupner and Brunel (2012)]. More recently, it has been suggested that in addition to local neuronal activity, certain global signals such as dopamine may also contribute to the learning process [Frémaux and Gerstner (2016); Bailey et al. (2000)].

Supervised learning in standard ANNs Given a predefined loss function with a minimum that solves the task at hand, the goal of learning is to adjust each weight (synapse) based on its overall impact on the value of loss. A first-order gradient descent method finds the local minumum of the loss function by iteratively updating weights in the direction opposite to the gradient. In this context, the popularization of the backpropagation algorithm [Rumelhart et al. (1986)] enabled to efficiently calculate the gradients for deep networks with a large number of parameters by propagating the error backwards through the network, see Fig. 1.4b. However, the error signal often decreases or increases exponentially with the number of backpropagated layers [Sussillo and Abbott (2014)], resulting in gradients that can be very close to zero or very large, i.e., *vanishing* or *exploding* gradients, respectively. The impact of the vanishing gradient problem [Bengio et al. (1994); Hochreiter et al. (2001)] is reduced network performance or failed training. To address this issue, standard ANNs often combine backpropagation with tricks of trade such as batch normalization and smart initialization [Goodfellow et al. (2016)].

Supervised learning in SNNs While biologically-inspired learning rules often struggle with training deeper networks, the effective backpropagation algorithm serves as inspiration for developing novel local learning rules that have the potential to scale better [Nøkland (2016); Akrout et al. (2019); Meulemans et al. (2021)]. In this thesis, we focus on supervised learning in spiking neural networks using the standard backpropagation algorithm, which can potentially later on be replaced with one of the local variants. For the purposes of training we consider two possible SNN representations: (i) *continuous-time* and (ii) *discrete-time*.

The continuous-time representation assumes a feed-forward artificial neural network where neurons communicate analog spiking times $t_j^{(n-1)}$ of the original spiking neural network. For the two networks to be equivalent, the ANN neuron has to follow a specific non-standard dynamics. In Fig. 1.5a, a particular integrate-and-fire neuronal dynamics was transformed into an ANN neuron where the output spiking time $t_i^{(n)}$ is calculated as a function of input spiking times $t_j^{(n-1)}$ that arrived before the threshold $\vartheta_i^{(n)}$ is reached [Zhang et al. (2021)]. The obtained ANN neuron dynamic is continuous in all situations, except when the input set of spiking times changes. For example, updating the spiking time $t_3^{(n-1)}$ to a later time instant in Fig 1.5a removes it from the input set and introduces a discontinuity. This could lead to issues when training such an ANN with backpropagation algorithm. Moreover, it is worth noting that while the mentioned tricks of the trade mitigate the vanishing gradient problem in standard ANNs, there is no guarantee that they work for the specific artificial neural networks introduced here.

The discrete-time representation assumes an SNN where neurons exchange spikes generated at one of the discrete times steps. The spike sent from neuron j in layer $n-1$ to neuron i in layer n is generated using a Heaviside step function. Training such a deep spiking neural network with the backpropagation algorithm has historically faced challenges due to the non-differentiable activation function, which makes gradient calculations not possible, see Fig. 1.5b. Several approaches have been attempted to address this issue: (i) injecting noise to smoothen out the activation function through a probabilistic approach [Gardner et al. (2015); Pfister et al. (2006)]; (ii) approximating the step function with a smooth deterministic function [Huh and Sejnowski (2018)] (iii) approximating the step function with a smooth deterministic

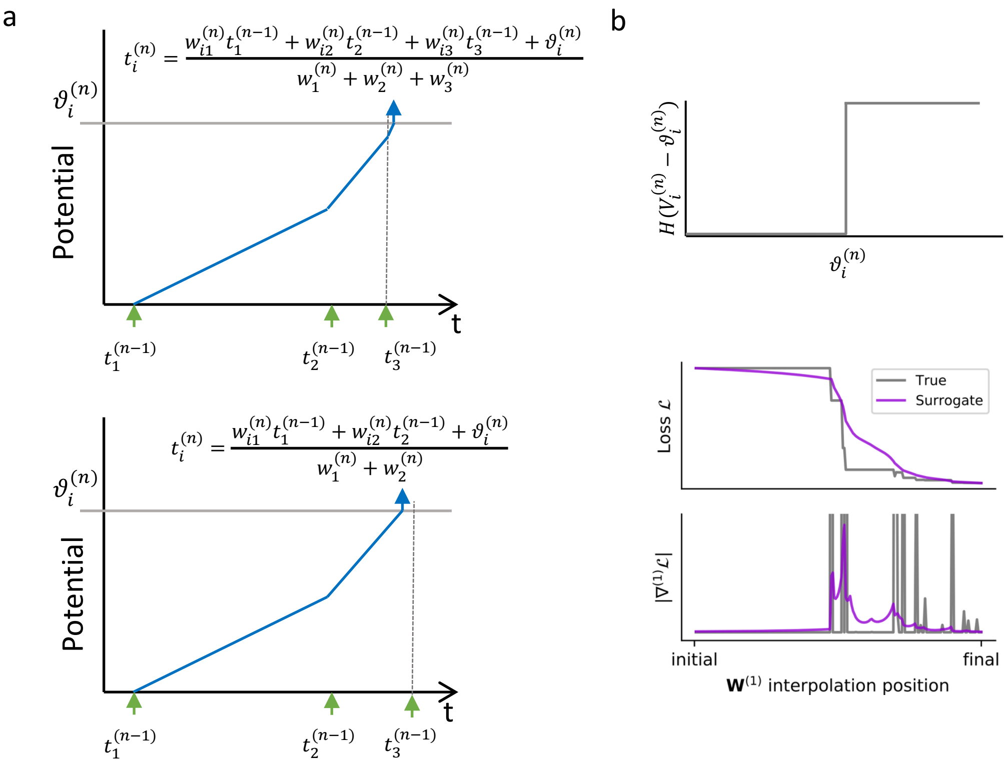


Figure 1.5: **Backpropagation in SNNs. a.** *Continuous-time representation.* An integrate-and-fire neuronal dynamics from SNN is transformed into a specific ANN neuron that calculates and outputs the analog spiking time of the SNN neuron. The non-linear transformation of the ANN neuron experiences discontinuities when an input spike $t_3^{(n-1)}$ (top) is removed from the input (bottom). This may cause a problem during training. **b.** *Discrete-time representation.* (top) Heaviside step function H that is used in the spike generation process. The derivative of this function is 0 everywhere except when the potential $V_i^{(n)}$ of neuron i in layer n is exactly equal to the neuron's threshold $\vartheta_i^{(n)}$ (i.e. $V_i^{(n)} = \vartheta_i^{(n)}$) and then its value is undefined; (bottom) The pseudo-derivative on the backward pass (purple) results in a significantly smoother loss, which facilitates the training. Reproduced from [Neftci et al. (2019)].

function, but only on a backward pass, i.e. the surrogate gradient approach [Neftci et al. (2019), Woźniak et al. (2020)], see Fig. 1.5b (bottom). Among these approaches, the surrogate gradient has demonstrated the most success in training deep spiking networks for complex tasks such as speech recognition [Bohnstingl et al. (2022a)].

1.4 SNNs with temporal coding

In this section we zoom in on spiking neural networks with temporal coding and explore potential methods to obtain high performance and sparse SNNs.

Converting a standard pretrained feed-forward ANN to an SNN A significant amount of prior research on SNNs has been devoted to converting pretrained standard artificial neural

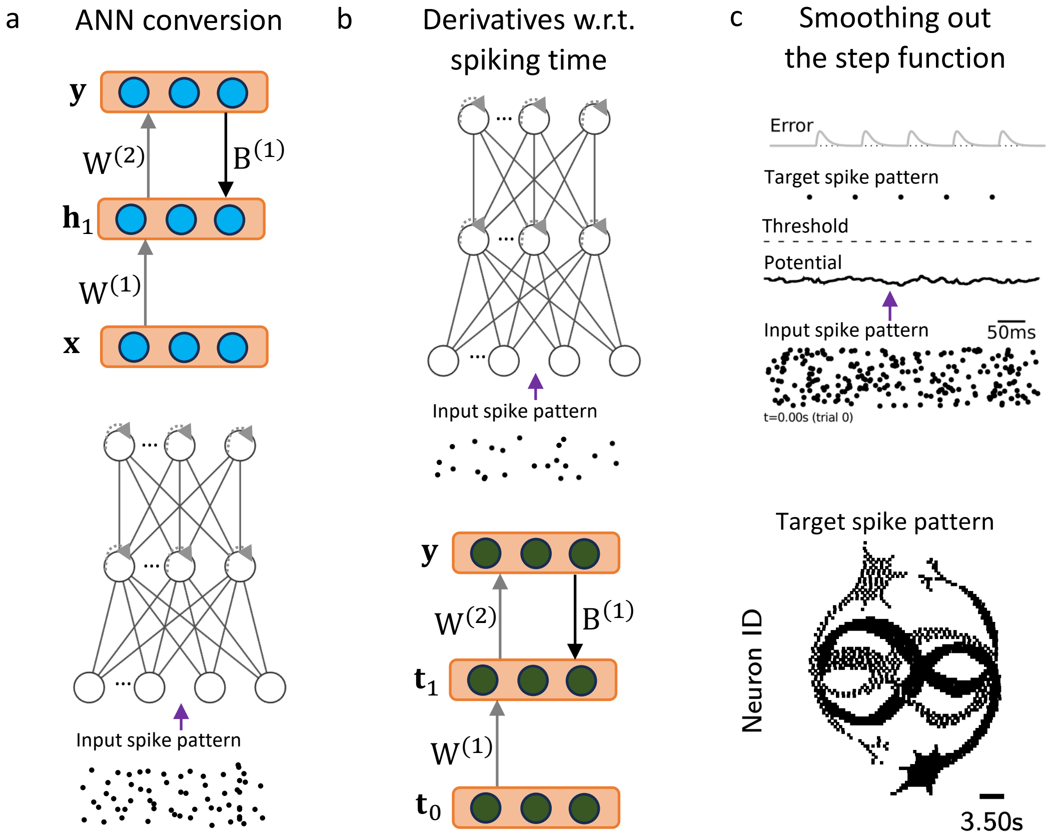


Figure 1.6: **SNNs with temporal coding.** **a.** A feed-forward ANN (top) with relatively standard neuron dynamics (light blue) can be converted into an SNN (bottom) with various temporal coding schemes and neuronal dynamics. **b.** *Continuous-time representation.* The TTFS coding scheme in SNNs often results in very sparse input. The SNN with TTFS coding scheme (top) can be transformed into a feed-forward ANN (bottom) where neurons communicate their spiking times and exhibit specific dynamics (dark green). **c.** *Discrete-time representation.* Learning to map a random input spatio-temporal spike pattern to a target spatio-temporal spike pattern using surrogate gradient. A more complex target spike pattern with multiple output neurons can also be learned (bottom). Reproduced from [Zenke and Ganguli (2018)].

networks into rate-coded spiking neural networks. More recently, there have been efforts to employ the conversion approach to temporally-coded SNNs (see Fig. 1.6a). [Rueckauer and Liu (2018)] pretrain a standard shallow ANN and convert it into an SNN with TTFS coding (see Fig. 1.2b), where neurons have dynamical threshold. Additionally, [Stockl and Maass (2021)] demonstrate that a neuron which spikes two times and has a complex parametrized dynamics can emulate a ReLU. Furthermore, [Bu et al. (2022)] recently proved that by selecting a specific activation function, it is possible to convert a deep ANN to a deep SNN without any error. Therefore, the conversion approach has the potential to yield deep and sparse neural networks with few spikes per neuron, while achieving performance comparable to ANNs on benchmark datasets. In Chapter 2 we explore the conversion of deep ReLU networks to deep SNNs with TTFS coding.

Supervised learning of SNNs with continuous-time representation In the pioneering work by [Bohte et al. (2002)], it was proposed to transform an SNN with TTFS coding into a feed-forward ANN in which neurons communicate their real-valued spiking times and exhibit specific dynamics (see Fig. 1.6b). The ANN is trained using the backpropagation algorithm where the derivatives are calculated with respect to the neuron's spiking time. Subsequent studies by [Mostafa (2018)] and [Zhang et al. (2021)] have shown that networks with a few hidden layers and various neuronal dynamics can be trained without any approximation. In Section A.1 we investigate methods to improve this optimization process further and bridge the remaining performance gap with ReLU networks. Moreover, the scalability of this approach to deep networks has not been well-established. In Chapter 3 we investigate the potential issues that may cause this limitation.

Supervised learning of SNNs with discrete-time representation The input vector of analog values and the target vector are converted into matrices of zeros and ones using one of the coding schemes (see Fig. 1.2b). An SNN that receives such spiking input is trained using backpropagation in conjunction with one of the smoothing methods. Assuming the spatio-temporal spike pattern coding, researchers such as [Gardner et al. (2015)] and [Zenke and Ganguli (2018)] have demonstrated that a shallow SNN can be trained to predict a specific target spike pattern when provided with an input spike pattern generated from random positions over fixed time span (see Fig. 1.6c). Section A.2 focuses on exploring the scalability of this approach to deeper networks and its applicability to real-world datasets. We investigate whether the training process which involves learning to map from random input spike patterns in the training data, can effectively generalize to unseen samples in the test data. Recently, it has also been demonstrated that SNNs with TTFS coding scheme can be effectively trained using one of the smoothing techniques [Cramer et al. (2022)].

1.5 Image classification task

Data classification is a fundamental task in traditional machine learning. The objective is to train a model that assigns a class (label) to a given input. The main metric reported is classification accuracy, which indicates the percentage of correct predictions on a given input set. Additionally, other metrics may also be considered, depending on the specific application and methodology. In this thesis, we primarily concentrate on image classification benchmarks. Some of the most significant ones include:

- MNIST and Fashion-MNIST [Lecun et al. (1998); Xiao et al. (2017)] consist of grayscale images. The MNIST images represent handwritten digits, while the Fashion-MNIST images contain items of clothing, see Fig. 1.7a. Each image is of size $28 \times 28 \times 1$ and there are 10 possible labels. Both datasets contain 60000 training and 10000 test images.

- CIFAR [Krizhevsky et al. (2009)] dataset consists of colored images featuring various objects, such as a car, see Fig. 1.7b. Each image is of size $32 \times 32 \times 3$ and can be classified

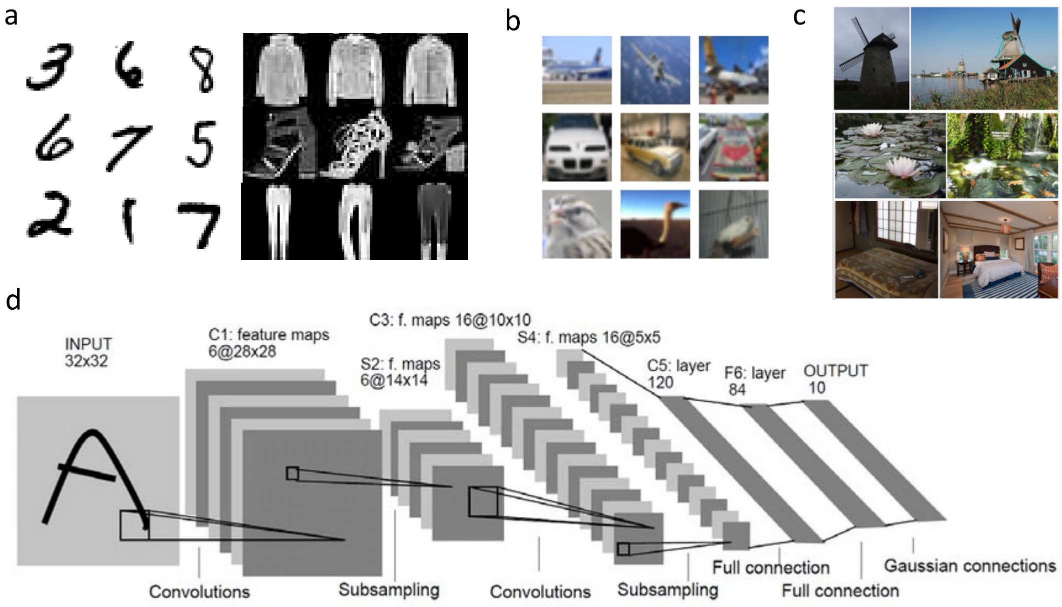


Figure 1.7: **Datasets and architectures.** **a.** Sample images from the MNIST (left) and the Fashion-MNIST (right) datasets. Reproduced from [Lecun et al. (1998) and Xiao et al. (2017)]. **b.** Sample images from the CIFAR dataset, which consists of colored images. Reproduced from [Krizhevsky et al. (2009)]. **c.** Sample images from the PLACES365 dataset, which consists of colored large-sized images. Reproduced from [Zhou et al. (2017)]. **d.** An example of a 5-layer convolutional neural network for image processing known as LeNet5. Reproduced from [Lecun et al. (1998)].

into 10 (CIFAR10) or 100 (CIFAR100) possible categories. The dataset includes 50000 training images and 10000 test images.

- PLACES365 [Zhou et al. (2017)] dataset is comprised of colored images depicting various scenes, such as a pond, see Fig. 1.7c. The size of each image can vary, but it is typically around 224 × 224 × 3, and there are 365 possible labels. The dataset contains 1.8 million training images, 36500 validation images and 328500 test images.

The PLACES365 dataset shares properties with the ImageNet dataset [Russakovsky et al. (2015); Yang et al. (2022)], which is omitted from our study due to data privacy issues.

Layers and architectures In general sense, a neural network typically consists of fully-connected layers, where each node in the current layer connects to all neurons in the subsequent layer. However, when it comes to image processing, additional layers and network architectures have shown to be beneficial. Convolutional layers, inspired by receptive fields in the visual cortex, allow each neuron to respond only to a limited part of the input, see Fig. 1.7d. To enable the application of the same filter across various patches, the synaptic strengths (weights) are shared among different neurons. Moreover, non-linear subsampling has proven to be a useful element of the processing pipeline. In the max-pooling layer, a neuron reduces the size of the input by replacing each patch with its maximal activity. The well-known LeNet5 architecture [Lecun et al. (1998)] exemplifies these essential components, see Fig. 1.7d.

1.6 Thesis structure

The thesis investigates brain-inspired spiking neural networks with sparse temporal coding with the idea of offering an energy-efficient alternative to standard ANNs. We explore supervised learning of SNNs as well as ANN-to-SNN conversion techniques in order to obtain high-performing spiking models for data classification tasks.

In Chapter 2, we focus on the neuronal dynamics with a linear postsynaptic response and explore SNN parameters that could establish an equivalence between deep SNN with time-to-first-spike (TTFS) coding and deep ReLU network. We investigate a family of potential exact mappings, where SNN parameters are determined as a function of weights and bias in a pretrained ReLU network. The significance of these mappings lies in the fact that an energy-efficient SNN can potentially replace the high-accuracy standard pretrained ANN without sacrificing performance. The primary contribution of this chapter is showing that a pretrained deep ReLU network, which may consist of fully-connected, convolutional, max pooling, or batch normalization layers, can be mapped to a mathematically equivalent SNN. As a result, we find that both networks exhibit the same performance on large image benchmarks, such as CIFAR100 and PLACES365.

In Chapter 3, our main focus is to explore whether the exact mappings from a deep ReLU network to a deep SNN could also lead to equivalent training of the two networks. We again start from the neuronal dynamics with a linear postsynaptic response and investigate the underlying reasons for the unsuccessful training of deep SNNs with TTFS coding. Additionally, we explore the conditions which are needed not only to perform the effective learning but to ensure that the training curves of the SNN and ReLU network align with each other. The main contribution of this chapter is the detection of the vanishing gradient problem in SNNs and an identification of an exact mapping that avoids this issue while also providing training equivalent to that of a standard feed-forward ANN. Consequentially, we discover that SNN can be trained to the same performance as ReLU network on CIFAR10, CIFAR100 and PLACES365, whereas the learning capability is crucial in mitigating the effects of quantization, noise or limited latency.

Chapter 4 summarizes our most important results and provides an outlook for future directions.

In Section A.1, our focus shifts to spiking neural networks with TTFS coding scheme, where predictions are allowed to be made before all the inputs are received. We are interested in the neuronal dynamics which enables translating a shallow SNN into a feed-forward ANN that closely approximates a ReLU network. Additionally, we explore the adaptation of batch normalization technique in this context with a goal of facilitating the backpropagation training.

The primary contribution of the section is enabling the training of our SNN by leveraging existing ReLU network techniques. We find that for MNIST dataset, our network is capable of low-latency classification while achieving the performance of a standard feed-forward ANN.

In Section A.2, two novel spatio-temporal spike pattern coding schemes are proposed to enable the generalization of learning to previously unseen data. We utilize backpropagation along with the surrogate gradient to train deep spiking neural networks and investigate the performance of such SNN classification systems on test data. Furthermore, we explore a multiplication-free implementation to further reduce the complexity of the system. The central contribution of this section are (i) correlative time encoding (CTE) and (ii) extended correlative time encoding (ECTE) schemes. We discover that by utilizing the optimal number of spikes, our network is able to perform a multiplication-free classification and achieve good performance on datasets such as MNIST and CIFAR10.

1.7 Author contributions

Chapter 2 is based on the [Stanojevic et al. (2022b)] paper. Ana Stanojevic and Guillaume Bellec conceived the idea. Ana Stanojevic and Wulfram Gerstner developed the theory. Ana Stanojevic designed and performed the simulations. Ana Stanojevic, Wulfram Gerstner, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini and Angeliki Pantazi analysed the results. Ana Stanojevic and Wulfram Gerstner wrote the manuscript with input from Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini and Angeliki Pantazi. The patent in preparation titled *Approximation-Free Mapping from Deep Artificial Neural Networks to Single-Spike Networks* by Stanojevic et al., directly relates to this work.

Chapter 3 is based on the [Stanojevic et al. (2023b)] paper. Wulfram Gerstner conceived the idea. Ana Stanojevic, Guillaume Bellec and Stanisław Woźniak developed the theory. Ana Stanojevic designed and performed the simulations. Ana Stanojevic, Wulfram Gerstner, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini and Angeliki Pantazi analysed the results. Ana Stanojevic and Guillaume Bellec wrote the manuscript with input from Wulfram Gerstner, Stanisław Woźniak, Giovanni Cherubini and Angeliki Pantazi.

Section A.1 is based on the [Stanojevic et al. (2022a)] paper. Wulfram Gerstner conceived the idea. Ana Stanojevic and Wulfram Gerstner developed the theory. Ana Stanojevic designed and performed the simulations. Ana Stanojevic, Wulfram Gerstner, Evangelos Eleftheriou, Giovanni Cherubini, Stanisław Woźniak and Angeliki Pantazi analysed the results. Ana Stanojevic wrote the manuscript with input from Wulfram Gerstner, Giovanni Cherubini, Stanisław Woźniak, Angeliki Pantazi and Evangelos Eleftheriou. The patent in preparation titled *Optimal Accuracy-Latency Trade-off in Single-Spike Spiking Neural Networks* by Stanojevic et al., is partially related to this work.

Section A.2 is based on the [Stanojevic et al. (2023a)] paper. Ana Stanojevic and Giovanni Cherubini conceived the idea and developed the theory. Ana Stanojevic designed and performed the simulations. Ana Stanojevic, Giovanni Cherubini, Stanisław Woźniak and Evangelos Eleftheriou analysed the results. Ana Stanojevic wrote the manuscript with input from Giovanni Cherubini, Stanisław Woźniak and Evangelos Eleftheriou. The patent titled *Correlative time coding method for spiking neural networks* [Cherubini, Stanojevic, and Sebastian (2022)] is partially inspired by this work.

2 An exact mapping from ReLU networks to spiking neural networks

Paper information

Authors: Ana Stanojevic, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini, Angeliki Pantazi, Wulfram Gerstner

Abstract Deep spiking neural networks (SNNs) offer the promise of low-power artificial intelligence. However, training deep SNNs from scratch or converting deep artificial neural networks to SNNs without loss of performance has been a challenge. Here we propose an exact mapping from a network with Rectified Linear Units (ReLUs) to an SNN that fires exactly one spike per neuron. For our constructive proof, we assume that an arbitrary multi-layer ReLU network with or without convolutional layers, batch normalization and max pooling layers was trained to high performance on some training set. Furthermore, we assume that we have access to a representative example of input data used during training and to the exact parameters (weights and biases) of the trained ReLU network. The mapping from deep ReLU networks to SNNs causes zero percent drop in accuracy on CIFAR10, CIFAR100 and the ImageNet-like data sets Places365 and PASS. More generally our work shows that an arbitrary deep ReLU network can be replaced by an energy-efficient single-spike neural network without any loss of performance.

Keywords spiking neural network, ReLU network, temporal coding, single-spike network, deep network conversion

Publication The chapter is under review at Neural Networks journal. arXiv version: ([Stanojevic et al. (2022b)])

Acknowledgements The research of W.G. and G.B. was supported by a Sinergia grant (No CRSII5 198612) of the Swiss National Science Foundation.

Declaration of competing interest. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

2.1 Introduction

Energy consumption of deep artificial neural networks (ANNs) with thousands of neurons poses a problem not only during training [Strubell et al. (2020); Patterson et al. (2022)], but also during inference [Brown et al. (2020)]. Among other alternatives [Hubara et al. (2016); Howard et al. (2017); Tan and Le (2019)], hardware implementations of biologically inspired spiking neural networks (SNNs) [Attwell and Laughlin (2001); Lennie (2003)] have been proposed as an energy-efficient solution [Burr et al. (2017); Sebastian et al. (2018); Göltz et al. (2020); Gallego et al. (2020); Göltz et al. (2021); Davies et al. (2021); Diehl et al. (2016)], not only for large centralized applications, but also for computing in edge devices [Wang et al. (2020); Boroumand et al. (2021); Jiang et al. (2018)]. In SNNs neurons communicate by ultra-short pulses, called action potentials or spikes, that can be considered as point-like events in continuous time. In deep multi-layer SNNs, if a neuron in layer n fires a spike, this event causes a change in the voltage trajectory of neurons in layer $n+1$. If, after some time, the trajectory of a neuron in layer $n+1$ reaches a threshold value, then this neuron fires a spike.

While there is no general consensus on how to best decode spike trains in biology [Rieke et al. (1996); Gerstner and Kistler (2002); Pillow et al. (2008)], multiple pieces of evidence indicate that immediately after an onset of a stimulus, populations of neurons in auditory, visual, or tactile sensory areas respond in such a way that the timing of the first spike of each neuron after stimulus onset contains a high amount of information about the stimulus features [Gollisch and Meister (2008); Johansson and Birznieks (2004); Kubke et al. (2002)]. These and similar observations have triggered the idea that, immediately after stimulus onset, an initial wave of activity is triggered and travels across several brain areas in the sensory processing stream [Optican and Richmond (1987); Thorpe et al. (1996, 2001); Hung et al. (2005); Yamins and DiCarlo (2016)]. We take inspiration from these observations and assume in this paper that information is encoded in the exact spike times of each neuron and that spikes are transmitted in a wave-like manner across layers of a deep feedforward neural network.

Specifically, we use coding by time-to-first-spike (TTFS) [Gerstner and Kistler (2002)], a timing-based code originally proposed in neuroscience [Gerstner (1998); Gerstner and Kistler (2002); Gollisch and Meister (2008); Johansson and Birznieks (2004); Thorpe et al. (2001)], which has recently attracted substantial attention in the context of neuromorphic implementations [Göltz et al. (2020); Gallego et al. (2020); Göltz et al. (2021); Rueckauer and Liu (2018); Comşa et al. (2021); Mostafa (2018); Zhang et al. (2021); Kheradpisheh and Masquelier (2020); Stanojevic et al. (2022a)]. In our implementation of TTFS coding, each neuron fires exactly one spike. The stronger the input to a given neuron the earlier it fires. Coding schemes with at most one spike per neuron are intrinsically sparse in terms of the number of spikes. Since spike generation is a costly process from the energetic point of view [Sorbaro et al. (2020)], TTFS coding paves the way towards implementations with low energy consumption.

While a relation between ReLU networks and networks of non-leaky integrate-and-fire neurons with TTFS coding has been suggested before [Rueckauer and Liu (2018); Zhang et al. (2021);

Kheradpisheh and Masquelier (2020); Mirsadeghi et al. (2021)], there has been so far a major obstacle that prevented a successful exact mapping from an arbitrary deep ANN to a deep SNN with TTFS coding. In a standard ANN, time is discretized and layers are processed one after the other. The hard problem of an exact conversion of ReLU activations into spike times arises from the fact that in an SNN spikes are point-like events that arrive asynchronously in continuous time. Imagine a neuron in a ReLU network that receives several positive inputs that add up to a value of 0.8 and several negative inputs that add up to a value of -1.0. Assuming a vanishing bias, the output of the unit is zero. However, if in the corresponding neuron of the SNN all the positive inputs have arrived *before* the negative inputs, the spiking neuron will have emitted a spike as soon as the firing threshold is reached [Rueckauer and Liu (2018)]. Yet, it is impossible to "call back" the spike later on so as to cancel it. A potential solution to this problem is to consider that each layer starts its computation only once the calculation in the previous layer has finished. In a recent conversion scheme a time-dependent threshold was used to enforce the necessary waiting time [Rueckauer and Liu (2018)]. However, due to imperfect conversion, there exists a small, but not negligible, loss in the final performance measure. Other conversion approaches for deep networks use custom activation functions [Bu et al. (2022)] for ANN training, or numerically optimized multi-spike codes for a given activation function [Stockl and Maass (2021)]. Moreover, most earlier approaches assumed that weights and biases of the ReLU network can be used as such in the corresponding SNN, but there is no fundamental reason why this should be the case.

What we would like for an exact mapping from ANN to SNN with TTFS coding is (i) a guarantee that no neuron fires too early so as to avoid the hard problem mentioned above; and (ii) a coding rule of how to translate the output $\bar{x}_i^{(n)}$ of neuron i in layer n of the ANN into a spike time of a corresponding neuron in the SNN. As an aside, we note that the the hard problem of TTFS coding disappears if the trajectory of spiking neurons is always positive. The ideal mapping approach starts from a standard ReLU network with or without convolutional layers, batch normalization, and max pooling and maps it by a potentially nonlinear transformation of parameters to a corresponding SNN without any performance loss using a well-defined mapping rule from rates to spike timings.

In this paper we construct an explicit mapping that addresses the points above and guarantees the mathematical equivalence of an SNN with the corresponding ANN. We assume that there is a pretrained ReLU network and that we have access to its weights and biases as well as the input data on which the network was trained. Using TTFS coding, we propose a conversion which maps a deep ANN with ReLUs to an equivalent deep SNN with non-leaky integrate-and-fire units without any loss in performance. In contrast to other methods which require fine-tuning of the SNN or training an SNN from scratch [Göltz et al. (2020); Kheradpisheh and Masquelier (2020); Neftci et al. (2019); Zenke and Ganguli (2018); Bohte et al. (2002); Tavanaei et al. (2019); Zenke and Vogels (2021); Bellec et al. (2020); Woźniak et al. (2020); Yan et al. (2021); Park et al. (2020)], the goal of our work is to derive the final SNN from the pretrained deep ReLU network. Further numerical approximation or optimization steps [Rueckauer and Liu (2018); Stockl and Maass (2021); Bu et al. (2022)] are not needed in our approach. The key to building an efficient TTFS conversion method is to derive an exact mathematical equivalence between an arbitrary deep ReLU network and the corresponding spiking network. Using standard pretrained models available online, we demonstrate conversion with 0% performance loss on CIFAR10 and CIFAR100 [Geifman (2018)] datasets as well as larger ImageNet-like [Russakovsky et al. (2015); Yang et al. (2022)] datasets such as Places365 [Zhou et al. (2017)] and PASS [Asano et al. (2021)] without any training or fine-tuning.

2.2 Results

The subsection 'Main Theoretical Result' formulates the precise claim of a family of exact mappings from ANN to SNN. We then present the main ideas of the proof for one specific mapping scheme before we sketch alternative mapping schemes. Finally we test our mapping algorithm on benchmark datasets.

2.2.1 Main Theoretical Result

Definition: Deep ReLU Network. *A Deep ReLU Network consists of* $M \geq 1$ *layers of hidden neurons with full or convolutional feedforward connectivity. Each neuron implements a rectified linear function* $x_i^{(n)} = [a_i^{(n)}]_+$, *where* $[]_+$ *denotes rectification, and its activation variable* $a_i^{(n)}$ *is defined as:*

$$a_i^{(n)} = \sum_j w_{ij}^{(n)} x_j^{(n-1)} + b_i^{(n)} \tag{2.1}$$

with weights $w_{ij}^{(n)}$ *and a bias* $b_i^{(n)}$. *An upper index* $n = 0$ *refers to the input layer and the i-th input is denoted with* $x_i^{(0)}$. *Optionally the network may also contain processing steps of max pooling and batch normalization.*

Our aim is to map each neuron of the Deep ReLU network to an integrate-and-fire neuron in the SNN so that each neuron fires exactly once. A spike at time $t_j^{(n-1)}$ of neuron j in layer $n-1$ generates a step current input with amplitude $J_{ij}^{(n)}$ into neuron i of layer n. The voltage trajectory of neuron i in layer n evolves according to

$$\frac{\mathrm{d}V_i^{(n)}}{\mathrm{d}t} = \alpha_i^{(n)} H(t - t_{\min}^{(n-1)}) + \sum_j J_{ij}^{(n)} H(t - t_j^{(n-1)}) + I_i^{(n)}(t) \tag{2.2}$$

where H denotes the Heaviside step function with $H(x) = 1$ for $x > 0$ or zero otherwise. The integration starts at time $t_{\min}^{(n-1)}$ with a slope $dV_i^{(n)}/dt = \alpha_i^{(n)} > 0$. The slope parameters $\alpha_i^{(n)}$, the weights $J_{ij}^{(n)}$, and the thresholds $\vartheta_i^{(n)}$ are parameters of the SNN. Neuron i in layer n may also receive an additional input $I_i^{(n)}(t)$. If the trajectory of $V_i^{(n)}$ crosses the threshold $\vartheta_i^{(n)}$ at time t from below, then $t = t_i^{(n)}$ is the firing time of neuron i in layer n. In our mapping, we use $I_i^{(n)}(t)$ to induce a short current pulse so as to trigger a spike at time $t_{\max}^{(n-1)}$ if neuron i has not fired before.

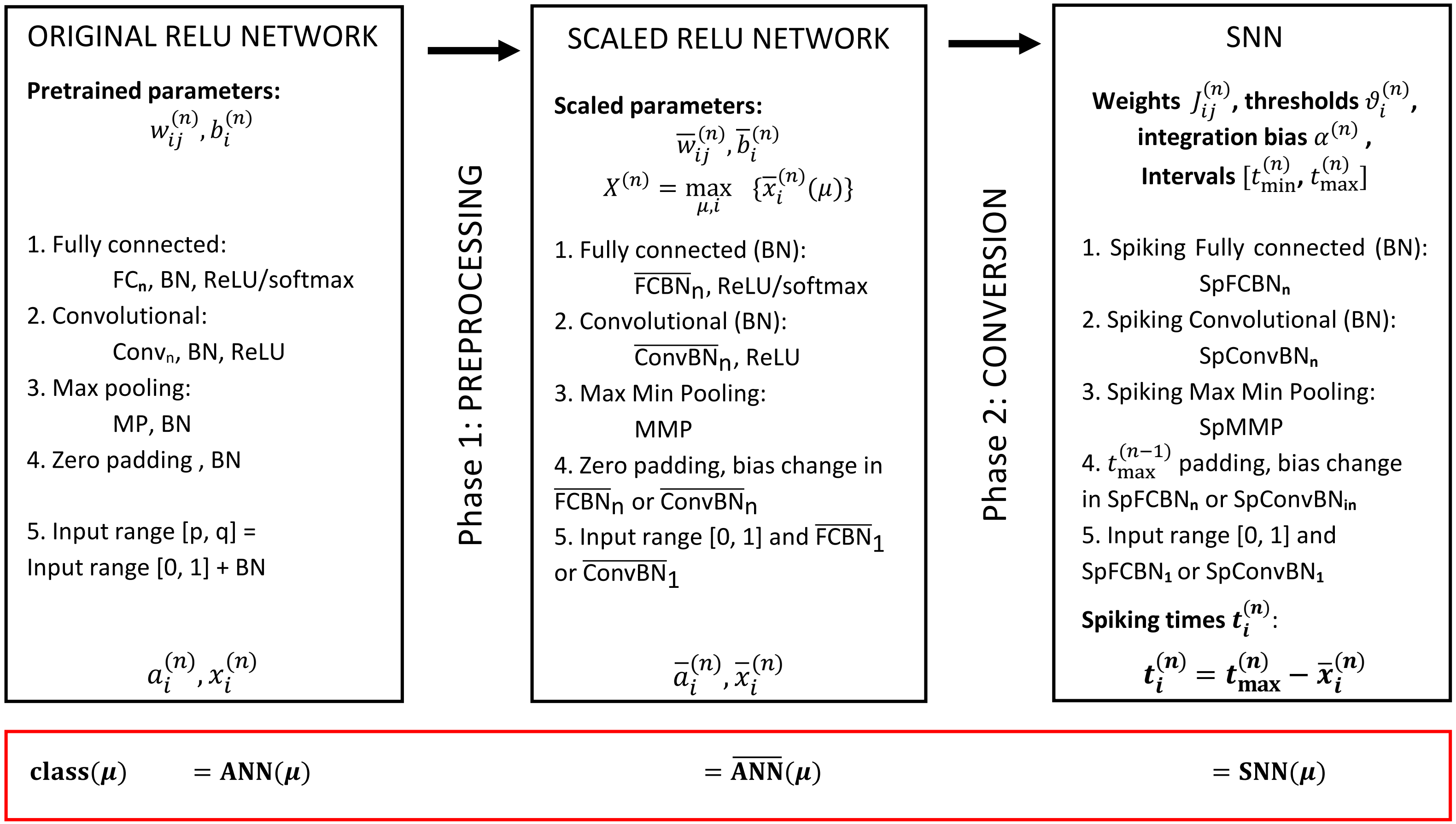

ORIGINAL RELU NETWORK
Pretrained parameters:
$w_{ij}^{(n)}, b_i^{(n)}$
1. Fully connected:
FC$_n$, BN, ReLU/softmax
2. Convolutional:
Conv$_n$, BN, ReLU
3. Max pooling:
MP, BN
4. Zero padding , BN
5. Input range [p, q] =
Input range [0, 1] + BN
$a_i^{(n)}, x_i^{(n)}$
Phase 1: PREPROCESSING
SCALED RELU NETWORK
Scaled parameters:
$\overline{w}_{ij}^{(n)}, \overline{b}_i^{(n)}$
$X^{(n)} = \max_{\mu,i} \{\overline{x}_i^{(n)}(\mu)\}$
1. Fully connected (BN):
$\overline{\text{FCBN}}_n$, ReLU/softmax
2. Convolutional (BN):
$\overline{\text{ConvBN}}_n$, ReLU
3. Max Min Pooling:
MMP
4. Zero padding, bias change in
$\overline{\text{FCBN}}_n$ or $\overline{\text{ConvBN}}_n$
5. Input range [0, 1] and $\overline{\text{FCBN}}_1$
or $\overline{\text{ConvBN}}_1$
$\overline{a}_i^{(n)}, \overline{x}_i^{(n)}$
Phase 2: CONVERSION
SNN
Weights $J_{ij}^{(n)}$, thresholds $\vartheta_i^{(n)}$,
integration bias $\alpha^{(n)}$,
Intervals $[t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}]$
1. Spiking Fully connected (BN):
SpFCBN$_n$
2. Spiking Convolutional (BN):
SpConvBN$_n$
3. Spiking Max Min Pooling:
SpMMP
4. $t_{\text{max}}^{(n-1)}$ padding, bias change
in SpFCBN$_n$ or SpConvBN$_{in}$
5. Input range [0, 1] and
SpFCBN$_1$ or SpConvBN$_1$
Spiking times $t_i^{(n)}$:
$t_i^{(n)} = t_{\text{max}}^{(n)} - x_i^{(n)}$
class(μ) = ANN(μ) = $\overline{\text{ANN}}(\mu)$ = SNN(μ)

Figure 2.1: Two phases of constructing a bidirectional mapping from ANN to SNN. In the first phase, Fully Connected (FC$_\text{n}$) or Convolutional (Conv$_\text{n}$) layers are identified; batch normalization (BN) is fused with the neighbouring layers yielding layers $\overline{\text{FCBN}}_\text{n}$ and $\overline{\text{ConvBN}}_\text{n}$, respectively; in case of zero padding, biases are adjusted at certain locations; max pooling steps are transformed to combined 'Max Min Pooling' (MMP) steps; and inputs are normalized to $[0,1]$. Then the scaling symmetry of ReLU $x_i^{(n)} = [a_i^{(n)}]_+ = C[a_i^{(n)}/C]_+$ for an arbitrary constant $C > 0$ is applied to bring the weights of the ReLU network into a desired range and the maximum output $X^{(n)}$ for each layer is calculated. Overline indicates scaled ReLU network. In the second phase, the resulting parameters $\{\bar{w}_{ij}^{(n)}, \bar{b}_i^{(n)}, X^{(n)}\}$ are mapped to the parameters $\{J_{ij}^{(n)}, \vartheta_i^{(n)}, \alpha^{(n)}, t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}\}$ of the SNN. For an arbitrary input data point μ the three networks have the same values in the output layer (before applying softmax) and are therefore predicting the same class(μ).

We claim that any Deep ReLU network can be mapped exactly to an SNN with integrate-and-fire neurons.

Theorem: Exact mapping from ANN to SNN. *Given the network parameters* $\{w_{ij}^{(n)}, b_i^{(n)}\}$ *of a Deep ReLU network that has been trained to high performance on a training set and given access to a representative subset of the input data of the training set, there exists a family of bidirectional mappings from the Deep ReLU network to an SNN with TTFS coding without any loss in performance where each ReLU is replaced by an integrate-and-fire unit with dynamics as in Eq. (2.2) and parameters* $\{J_{ij}^{(n)}, \vartheta_i^{(n)}, \alpha_i^{(n)}, t_{\min}^{(n)}, t_{\max}^{(n)}\}$.

Remarks. (i) The theorem mentions a family of mappings since the mapping is not unique, i.e., different combinations of parameters in the SNN give rise to an exact mapping. (ii) In the family of mappings that we consider each neuron emits at most a single spike. (iii) A consequence of the exact mapping is that both SNN and ANN have exactly the same performance on a sample-by-sample basis: if for a specific sample the prediction of the ANN is wrong then this is also the case for the SNN, and vice versa. (iv) One of the potential mappings is such that slope parameter $\alpha_i^{(n)}$ is identical for all neurons and all layers as stated in the following corollary.

Corollary: Mapping with fixed α. *An Exact mapping from ANN to SNN is possible with a slope parameter* $\alpha_i^{(n)} = \alpha > 0$ *that is identical for all neurons in all layers. Moreover, we may choose* $t_{\min}^{(n)} = t_{\max}^{(n-1)}$.

2.2.2 Proof Sketch of Main Theoretical Result

Our proof is constructive, i.e., we propose an explicit mapping algorithm. The arguments work for arbitrary $\alpha_i^{(n)} > 0$. At the end of the argument we set $\alpha_i^{(n)} = \alpha$ to instantiate the conditions of the Corollary; see Methods for details. The algorithm has two phases, see Fig. 2.1.

Phase 1: Preprocessing. The original ReLU network undergoes preprocessing such that the network expects input in the $[0, 1]$ range and without any performance loss; furthermore batch normalization steps are removed by fusing them with the weights of neighbouring layers; see Fig. 2.2, Algorithm 1 and Methods for details.

Importantly, and different to other studies in the field of network conversion, we use the known scaling symmetry of the ReLU activation function, i.e., $[a_i^{(n)}]_+ = C[a_i^{(n)}/C]_+$ for an arbitrary constant $C > 0$, to implement a nonlinear transformation from the original weight and bias parameters of the ReLU network to new parameters $\{\bar{w}_{ij}^{(n)}, \bar{b}_i^{(n)}\}$. After the transformation we can guarantee that the sum of the weights in each neuron is bounded in the range $(-B_{\text{low}}) \leq \sum_j \bar{w}_{ij}^{(n)} \leq 1 - \delta$ for hyperparameters $B_{\text{low}} > 0$ and $0 < \delta \ll 1$. The transformation proceeds layer-wise from the input to the output layer and does not change the network output. Neuron

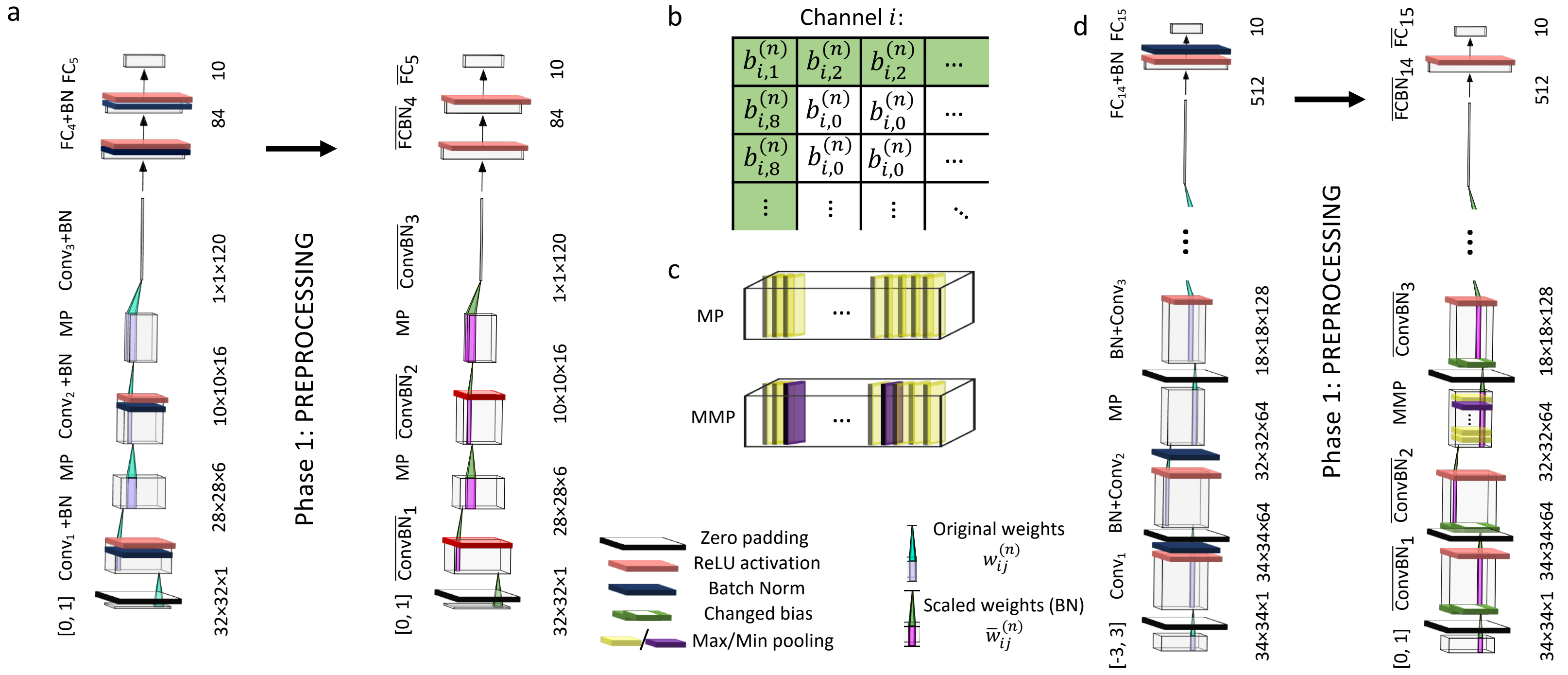


Figure 2.2: **Preprocessing** for LeNet5 and VGG16 networks: **a.** LeNet5 original ReLU network with batch normalization (BN, black rectangular sheet) before the activation function (red sheet); during preprocessing batch normalization is fused with *previous* convolutional (Conv_n) and fully connected (FC_n) layers. **b.** When fusing a batch normalization layer with the next convolutional layer containing zero padding, some of the biases (in green) are changed; **c.** When fusing batch normalization with the next convolutonal layer with max pooling (yellow) in between, specific channels might be changed to use min pooling function (violet). **d.** VGG16 original ReLU network used for CIFAR10 classification with batch normalization after the activation function; during preprocessing batch normalization is fused with *following* convolutional and fully connected layers.

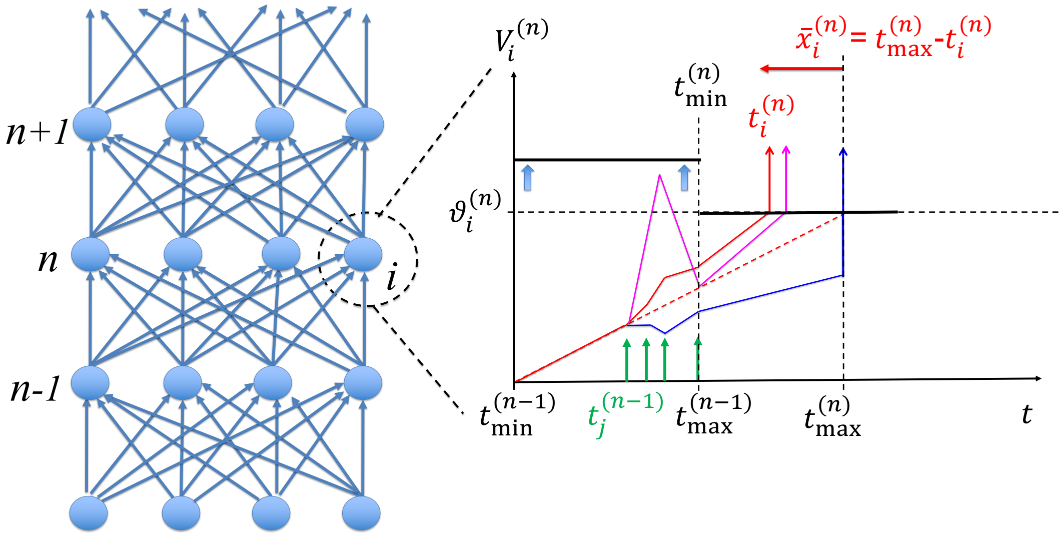


Figure 2.3: **Mapping** to multi-layer SNN with TTFS encoding and integrate-and-fire units. **Left:** Neurons in layer $n-1$ are connected to neuron i in layer n. **Right:** Spikes from neurons $1 \leq j \leq N^{(n-1)}$ arrive at times $t_j^{(n-1)}$ (green vertical arrows). The red trajectory shows the evolution of the potential $V_i^{(n)}$ as a function of time. Neuron i fires at time $t_i^{(n)}$ (red vertical arrow) when $V_i^{(n)}$ reaches the threshold $\vartheta_i^{(n)}$. The output value $\bar{x}_i^{(n)}$ of the corresponding ReLU corresponds to the time difference between $t_i^{(n)}$ and $t_{\max}^{(n)}$. Other neurons in layer n fire at other moments (blue, magenta). No neuron in layer n can fire later than $t_{\max}^{(n)}$. Our exact mapping procedure guarantees, that for all neurons the slope of the trajectory at the moment of spike firing is positive. Moreover, since in our theorem the threshold is arbitrarily high (thick blue vertical arrows) for $t < t_{\min}^{(n)}$, all firings in layer n occur for $t > t_{\min}^{(n)}$ so that the hard problem of late inhibitory input is solved.

i in layer n of the scaled ReLU network has an activation variable $\bar{a}_i^{(n)}$ and output $\bar{x}_i^{(n)}$. The mapping is bidirectional so given the quantities $\bar{a}_i^{(n)}$ and $\bar{x}_i^{(n)}$ of the scaled network we can recover the variables $a_i^{(n)}$ and $x_i^{(n)}$ of the original network; see Methods Eqs. (2.20)-(2.23). Finally, since we have access to a representative sample of input data used during training, we extract $X^{(n)}$, the maximal activation of the rescaled ReLUs in layer n, across the input data and all neurons in layer n.

Phase 2: Conversion. To construct an exact conversion of the scaled ReLU network to the network of spiking neurons without any loss in performance we exploit six essential ideas (see Methods for details):

(i) *Choice of TTFS code*. We construct a mapping such that each neuron i in layer n of the SNN emits exactly one spike at $t_i^{(n)}$, where $t_{\min}^{(n)} < t_i^{(n)} \leq t_{\max}^{(n)}$ (see Fig. 2.3). Positive activation leading to a ReLU output $\bar{x}_i^{(n)} = \bar{a}_i^{(n)} > 0$ corresponds to an *early* firing time $t_i^{(n)} = t_{\max}^{(n)} - \bar{a}_i^{(n)}$, or equivalently, $\bar{a}_i^{(n)} = t_{\max}^{(n)} - t_i^{(n)}$. Thus, spike times depend *linearly* on the output $\bar{x}_i^{(n)}$ of active ReLU neurons. Moreover if a neuron in layer n has not fired before time $t_{\max}^{(n)}$, it receives an additional external input pulse $I_i^{(n)}(t) = R\,\vartheta_i^{(n)}\,\delta(t - t_{\max}^{(n)})$ with $R \gg 1$ that triggers immediate firing at time $t_{\max}^{(n)}$. The parameters $\alpha_i^{(n)}$, $\vartheta_i^{(n)}$, the times $t_{\min}^{(n)} = t_{\max}^{(n-1)}$, as well as the weights $J_{ij}^{(n)}$ of the spiking network are determined during the conversion for all i, j, n as described in Methods (see Algorithm 2) and are kept fixed thereafter. With this coding scheme each neuron in layer $n-1$ fires exactly once up to $t_{\max}^{(n-1)} = t_{\min}^{(n)}$. Therefore for $t > t_{\min}^{(n)}$ all input spikes to neurons in layer n have arrived.

(ii) *Slope of trajectory.* Since for $t > t_{\min}^{(n)}$ all input spikes to neurons in layer n have already arrived, the trajectory of neuron i in layer n has, a *constant* slope $dV/dt = \sum_j J_{ij}^{(n)} + \alpha^{(n)}$ [see. Eq. (2.2)] which is *independent* of the sequence of spike arrivals.

(iii) *Weight conversion.* Since for $t > t_{\min}^{(n)}$ the trajectories have constant slope, the mapping from activations in the ReLU to firing times in the SNN can be derived from the threshold-crossing condition $V_i^{(n)} = \vartheta_i^{(n)}$ for each neuron i in layer n. Evaluating this condition yields the nonlinear conversion of weights

$$J_{ij}^{(n)} = \frac{\alpha_i^{(n)}}{1 - \sum_{j'} \bar{w}_{ij'}^{(n)}}\, \bar{w}_{ij}^{(n)} \tag{2.3}$$

which is invertible. A similar invertible relation holds for the bias parameter (see Methods). Thus weights in the scaled ANN can be mapped to weights in the SNN without sign change. Summation over j on both sides of Eq. (2.3) shows that the slope has a value $\alpha_i^{(n)} + \sum_j J_{ij}^{(n)} = \alpha_i^{(n)}/(1 - \sum_j \bar{w}_{ij}^{(n)}) > 0$. Thus, once all input spikes have arrived, the slope of the trajectories is positive because of the weight rescaling $\sum_j \bar{w}_{ij}^{(n)} < 1$; This is the key motivation for the weight rescaling in Phase 1.

(iv) *Choice of* $t_{\max}^{(n)}$. Given our TTFS code, we know that a stronger activation leads to earlier spikes, yet we have to make sure that no neuron in layer n fires before the last spike of neurons in layer $n-1$. The earliest possible spike in layer n occurs at time $t_{\max}^{(n)} - X^{(n)}$ where $X^{(n)}$ is the maximal activation of ReLU neurons in layer n identified during the preprocessing phase. We therefore set $t_{\max}^{(n)} = t_{\min}^{(n)} + (1+\zeta)X^{(n)}$, where $\zeta > 0$. In practice (see below) a value of $\zeta = 0.5$ works well.

(v) *Choice of threshold.* By definition of our TTFS code, $t_{\max}^{(n)}$ is the time when a neuron in layer n that corresponds to a ReLU with activation $\bar{a}_i^{(n)} = 0$ reaches the threshold $\vartheta_i^{(n)}$; therefore this condition defines the value of the threshold. Because of different biases and different weights for different neurons, the thresholds $\vartheta_i^{(n)}$ are neuron-specific (see Methods). *This finishes the proof in the general case.*

(vi) *Free slope parameter.* Since the slope factor $\alpha_i^{(n)}$ is a free parameter, we can arbitrarily set $\alpha_i^{(n)} = 1$ for all neurons across all layers $1 \le n \le M$. This yields the Corollary. The condition of the Corollary is the specific case used in the simulations.

Remark. We may wonder how the above points solve the hard problem of TTFS coding. Our analysis above makes no statement about the trajectories of neurons in layer n for the time $t < t^{(n)}_{\text{min}}$. Therefore we formally set the threshold for $t < t^{(n)}_{\text{min}}$ to an arbitrarily high value to ensure that no spike occurs before $t^{(n)}_{\text{min}}$. As mentioned under point (iii), our method guarantees a positive slope *after* $t^{(n-1)}_{\text{max}} = t^{(n)}_{\text{min}}$. Since during the allowed spiking interval $[t^{(n)}_{\text{min}}, t^{(n)}_{\text{max}}]$ the slope is fixed and positive, a spike never needs to be "called back". Because of the preprocessing, we know that $\sum_j \bar{w}^{(n)}_{ij} > -B_{\text{low}}$. For example, a choice $B_{\text{low}} = 10$ and $\alpha^{(n)} = 1$ yields a slope larger than $1/11$. Furthermore, because of our choice of $t^{(n)}_{\text{max}}$ under point (iv) we know that the interval $[t^{(n)}_{\text{min}}, t^{(n)}_{\text{max}}]$ is long enough to allow even the most activated neuron to fire at the correct time. Finally, because of our choice of TTFS code under point (i) we are sure that all neurons in layer $n-1$ have fired before or at $t^{(n)}_{\text{min}}$. These choices together solve the hard problem of TTFS coding.

2.2.3 Examples of equivalent mappings

As stated in the main Theorem, the mapping from ANN to SNN is not unique; rather there is a family of equivalent mappings. Here we present several concrete implementation schemes.

Mapping with guaranteed positive slope

In the proof sketch above it was shown that the slope of all neurons is always positive once all input spikes have been received. However, we cannot exclude that *before* the time $t^{(n)}_{\text{min}}$ the trajectory transiently has a negative slope; see Fig. 2.3. If this is desired for some application, we can use the free parameter $\alpha^{(n)}_i = \alpha^{(n)}$ to ensure that the slope of the trajectory is *always* non-negative, even before $t^{(n)}_{\text{min}}$. To do so, we sum over all negative weights incoming to a given neuron and choose the slope parameter in layer n such that

$$\alpha^{(n)} + \min_i \sum_j J^{(n)}_{ij} H(-J^{(n)}_{ij}) > 0 \tag{2.4}$$

This ensures that the slope is positive not only if all inhibitory spikes arrive before the first excitatory spike, but also for all other possible timings of inhibitory input spikes. Therefore the hard problem of late inhibitory spikes can even be solved with a threshold that remains *constant* throughout the processing, i.e., even before $t^{(n)}_{\text{min}}$. In practice we found that we could work with a constant threshold even if we did not implement the strict condition on the slope parameter formulated in Eq. (2.4) but worked instead with $\alpha^{(n)} = 1$. The strict condition in Eq. (2.4) can lead to very large slope parameters $\alpha^{(n)}$ and high firing thresholds $\vartheta^{(n)}_i$ which we might want to avoid in hardware implementations because of increased energy consumption.

Mapping with a dynamical threshold

In the proof sketch we assumed a constant threshold $\vartheta_i^{(n)}$ for all times $t > t_{\min}^{(n)}$. However, we can reinterpret the slope factor as a dynamical threshold. To see this, we integrate Eq. (2.2) and write the threshold condition that determines the firing time $t_i^{(n)} < t_{\max}^{(n)}$ in the form

$$\vartheta_i^{(n)} = V_i^{(n)}(t_i^{(n)}) = \alpha_i^{(n)}[t_i^{(n)} - t_{\min}^{(n-1)}] + \sum_j J_{ij}^{(n)} \epsilon(t_i^{(n)} - t_j^{(n-1)}) \tag{2.5}$$

where we have suppressed the external input and $\epsilon(s)$ is the voltage response to an input spike arriving at $s = 0$ [Gerstner and Kistler (2002)]. Using standard textbook arguments, the term $\alpha_i^{(n)}[t_i^{(n)} - t_{\min}^{(n-1)}]$ can be shifted to the left-hand side which gives rise to a 'dynamical threshold' [Gerstner and Kistler (2002)] defined as $\vartheta_i^{(n)}(t) = \vartheta_i^{(n)} - \alpha_i^{(n)}[t - t_{\min}^{(n-1)}]$. Thus, the mapping in the corollary is identical to a mapping where the slope factor vanishes, but each neuron has a dynamical threshold that decreases linearly with time.

Mapping with identical weights in SNN and ANN

Previous studies have proposed approximative mappings under the condition $J_{ij}^{(n)} = \bar{w}_{ij}^{(n)}$ for all neurons in all layers. A quick glance at Eq. (2.3) tells us that a mapping with $J_{ij}^{(n)} = \bar{w}_{ij}^{(n)}$ becomes exact under the condition of a neuron-specific slope parameter

$$\alpha_i^{(n)} = 1 - \sum_j \bar{w}_{ij}^{(n)}. \tag{2.6}$$

Thus, in contrast to the mapping in the proof sketch of the Theorem, the slope parameter $\alpha_i^{(n)}$ is no longer a free parameter but *must* be chosen according to Eq. (2.6) if the aim is to have the same set of weights in ANN and SNN. Interestingly, under this condition, the trajectory of all neurons have the same slope of value one for $t > t_{\min}^{(n)}$.

Mapping with less than one spike per neuron

Even though our theory requires each neuron to spike exactly once, it is possible to have an alternative implementation where a given spiking neuron fires only when the corresponding ReLU is *active*. Instead of sending (costly) spikes of inactive neurons, it is sufficient to store the reference times $t_{\max}^{(n)}$ for all n. The trick is to set the slope of all trajectories of neurons i in layer n to $\alpha_i^{(n)} + \sum_j J_{ij}^{(n)}$ as soon as the maximum spike time $t_{\max}^{(n-1)}$ of neurons in layer $n-1$ has been reached. This is mathematically equivalent to making all inactive neurons in layer $n-1$ fire at time $t_{\max}^{(n-1)}$ but reduces the overall number of spikes in the network significantly. Therefore each neuron fires *at most* one spike. Since the ANN implements a nonlinear function from input to output, at least one neuron has to be inactive for at least one input data point, so that we know that on average there is strictly less than one spike per neuron. Since we are interested in a low-energy solutions, we report in the following the average number of active neurons across all inputs and all neurons for the given dataset. This number can be interpreted as 'spikes per neuron per classification'. Note that this number depends on the specific regularization used during training of the ANN and can be further reduced by appropriate loss functions. Examples of this approach will be given below.

2.2.4 Performance on Benchmark Datasets

The above algorithm is a constructive proof that an exact mapping is possible. However, it is not clear how well it would perform in practice since there might be stability issues in the implementation or long processing delays that would reduce the attractivity of the mapping. In the following we test this algorithm on several image classification tasks with different standard datasets.

For each data set, we report the classification accuracy for the original ReLU network, for the SNN, as well as the percentage of agreement on a image-by-image basis between class prediction of original ReLU network and SNN network. Agreement is 100 percent, if for each image that is correctly (wrongly) classified by the ReLU, the image is also correctly (wrongly) classified by the SNN. Furthermore we report percentage of spikes per neuron under the implementation scheme mentioned at the end of the previous subsection.

MNIST, Fashion-MNIST and CIFAR10

In order to compare our results with existing conversion approaches [Rueckauer and Liu (2018); Yan et al. (2021)], we include MNIST and Fashion-MNIST [Deng (2012); Xiao et al. (2017)] as well as CIFAR10 [Geifman (2018)] in our evaluation. We consider 16-layer VGG16, 5-layer LeNet5 and 2-layer fully connected networks, see Table 2.1. VGG16 contains max pooling, fully connected and convolutional layers together with zero padding and batch normalization applied after ReLU activation functions. For the MNIST dataset the SNN achieves the same 99.6% accuracy as the original ReLU network with 100% agreement, whereas the number of active neurons is around 51%. Similarly, for Fashion-MNIST there is a 100% agreement between SNN and ReLU predictions with the accuracy of 93.7% and around 45% of active neurons. In [Rueckauer and Liu (2018)] the authors perform a conversion of a 2-layer fully connected network as well as a LeNet5 convolutional network such that the weights and biases of the SNN and ANN are identical. We reproduce the ANN results of those models and compare the performance of their SNN with the one obtained using our method. Our SNN surpasses the accuracy in [Rueckauer and Liu (2018)], and has 100% agreement between SNN and ANN with around 50% active neurons. Moreover, if we apply L1 activity regularization to the 2-layer fully connected network, MNIST images are classified with the accuracy of 98.52% with only 20 spikes in the hidden layer, i.e. 3.33% of active neurons. The original and the scaled LeNet5 network can be seen in Fig. 2.2a. For the preprocessing and mapping details refer to the Methods.

Model & dataset	Image size	Classes	Accuracy [%]		Agreement [%]	Spikes [%]
			ReLU	SNN		
Fully connected, MNIST [Rueckauer and Liu (2018)]	28 × 28 × 1	10	98.50	98.35	-	-
Fully connected, MNIST [ours]	28 × 28 × 1	10	98.52	**98.52**	100	50.28
Fully conn. L1, MNIST [ours]	28 × 28 × 1	10	98.52	**98.52**	100	**3.33**
LeNet5, MNIST [Rueckauer and Liu (2018)]	28 × 28 × 1	10	98.96	98.57	-	-
LeNet5, MNIST (Fig. 2.2a) [ours]	28 × 28 × 1	10	99.03	**99.03**	100	50.18
VGG16, MNIST [ours]	28 × 28 × 1	10	99.60	99.60	100	51.21
VGG16, Fashion-MNIST [ours]	28 × 28 × 1	10	93.70	93.70	100	45.34
VGG16, CIFAR10 [Yan et al. (2021)]	32 × 32 × 3	10	92.55	92.48	-	-
VGG16, CIFAR10 (Fig. 2.2d) [ours]	32 × 32 × 3	10	93.59	**93.59**	100	38.38
VGG16 L1, CIFAR10 [ours]	32 × 32 × 3	10	93.16	93.16	100	**21.51**
Large-scale tests						
VGG16, CIFAR100 [ours]	32 × 32 × 3	100	70.48	70.48	100	38.21
VGG16, Places365 [ours]	224 × 224 × 3	365	52.69	52.69	100	53.72
VGG16, PASS [ours]	224 × 224 × 3	1000	N/A	N/A	100	53.24

Table 2.1: **Comparing performance of original ReLU networks and SNNs**. Agreement metric shows percentage of inputs for which original ReLU network and SNN network predict the same class. Spikes column reveals the average percentage of active neurons across all hidden layers when the mapping with less than one spike per neuron is applied. Places 365 and PASS are used as substitutes for ImageNet which breaches data privacy [Yang et al. (2022)]. Accuracy calculation is not applicable for PASS dataset, as it is unlabeled. L1 denotes model with regularization. The highest accuracy and the lowest percentage of spikes for model/dataset pairs across different methods are highlighted in bold.

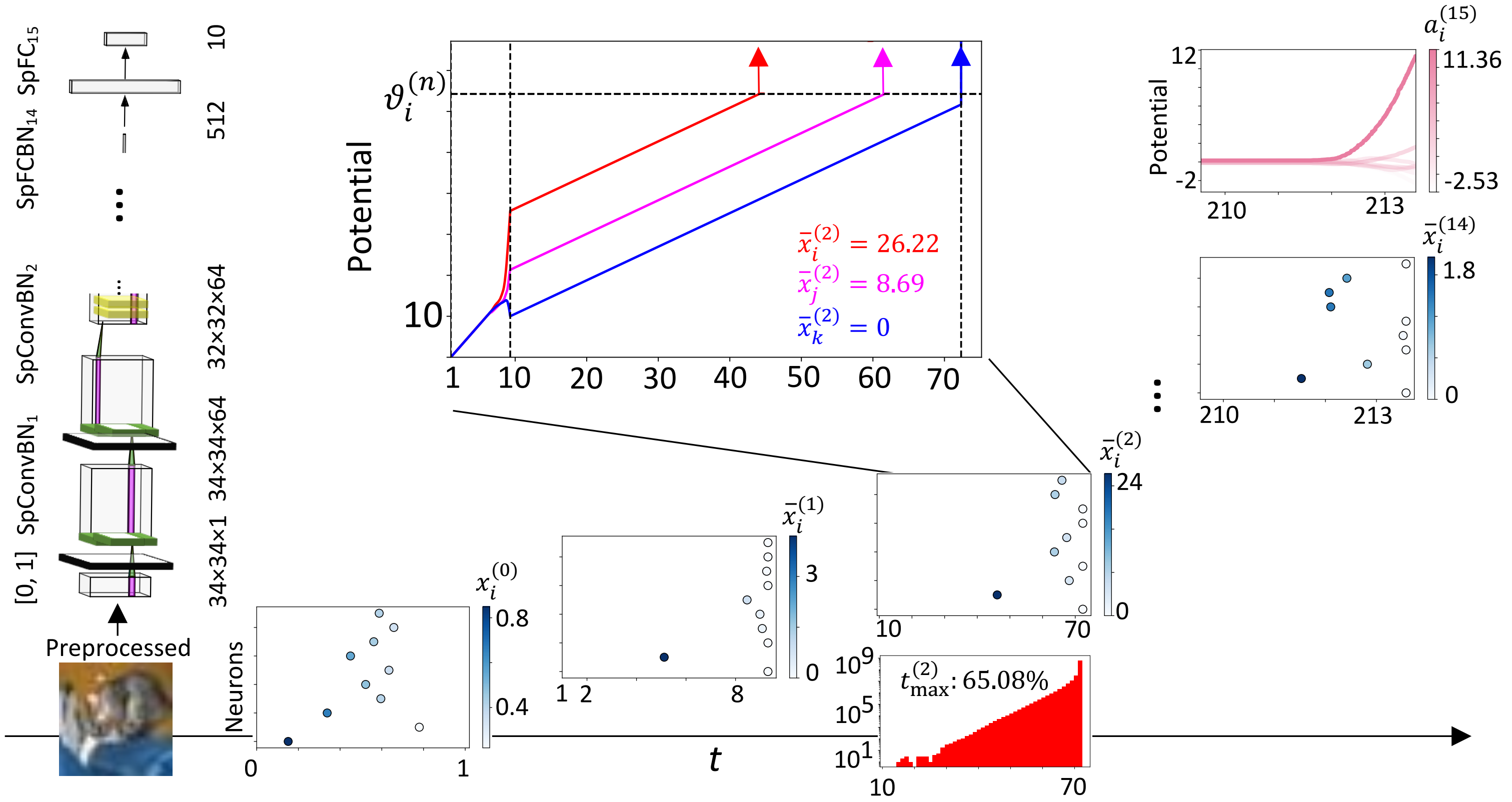


Figure 2.4: **Image Classification with spikes**. A presentation of a cat image from the CIFAR dataset triggers spikes (filled and open circles) in consecutive layers of the SNN. For layers 0, 1, 2 and 14, one of the neurons is the one that fires the earliest spike for this image whereas the other nine are randomly selected. In a given layer, earlier spike times correspond to larger values (darker color) of the corresponding ReLU in the original network. At the output layer, the maximum potential corresponds to the largest activation variable in the ReLU network. **Zoom inset:** Voltage trajectories of three neurons from the same convolutional channel in layer 2. Spike times correspond to the moments of threshold crossing. **Histogram inset:** Histogram of spike counts per time bin of neurons in layer 2 averaged across all neurons and all images in the test set. Only 34.92% of neurons fire before $t_{\max}^{(2)}$.

CIFAR10 contains color images of ten classes. The pretrained weights were obtained from an online repository [Geifman (2018)] where a convolutional network similar to the VGG16 architecture (see Table 2.1) was used. It comprises 15 layers in total since it uses only two fully connected layers instead of three (Fig. 2.2d). The network contains max pooling and convolutional layers together with zero padding and batch normalization applied after the ReLU activation functions. For CIFAR10 the SNN achieves the same 93.59% accuracy as the original ReLU network with 100% agreement between the two networks, whereas at 38% the number of active neurons is smaller than for the other datasets. If we apply L1 activity regularization, a small subset of 21.51% active hidden neurons is enough to perform classification with 93.16% accuracy. Further reduction of the number of spikes to 15.52% already decreases the accuracy to 90.54%.

In Fig. 2.4 we show an example of an SNN inference for classification of a cat image from CIFAR10. For input and hidden layers a raster plot of 10 neurons is shown and the spikes of neurons with higher activation of the corresponding neuron in the ANN are color-coded with darker shade. At the input layer the value of the data can be recovered from the spiking time of neuron i as $x_i^0 = 1 - t_i^{(0)}$ and in layer n the output of a neuron i of scaled ReLU network can be recovered from the spiking time of the corresponding neuron in the SNN as $\bar{x}_i^{(n)} = t_{\max}^{(n)} - t_i^{(n)}$. The duration of the interval $[t_{\min}^{(n)}, t_{\max}^{(n)}]$ varies considerably from one layer to the next. At the output layer $n = 15$ a potential with darker color indicates a larger value of the activation variable of the corresponding neuron in the original ReLU network. At time $t_{\max}^{(15)} = t_{\max}^{(14)} + 0.1$ when all the input from the layer $n = 14$ has arrived, the maximum potential corresponds to the neuron with maximal activation variable, i.e. both networks predict the same class.

Large-scale data sets

We avoided the ImageNet dataset because of privacy-concerns [Yang et al. (2022)] and used instead Places 365, PASS, and CIFAR100 for more realistic tests. The 'Places365-Standard' dataset contains high-resolution color images [Zhou et al. (2017)] resembling those in the ImageNet dataset. The pretrained weights are obtained from an online repository [Zhou (2018)] that contains a standard VGG16 network without batch normalization which we map to a corresponding SNN; see Table 2.1. The SNN achieves the same 52.69% accuracy as the original ReLU network with 100% agreement between the two networks, whereas the number of active neurons is around 53%.

The PASS dataset consists of 1.4 million unlabeled images [Asano et al. (2021)] and is used as a substitute for ImageNet [Asano et al. (2021)] so as to avoid privacy-concerns. We use the same network as for the 'Places365-Standard' dataset, see Table 2.1. The weights are downloaded from the VGG16 model for ImageNet available in TensorFlow [Abadi et al. (2015); Russakovsky et al. (2015)]. An inference on an image from PASS returns one of the 1000 ImageNet classes as output. When performing inference with the SNN we verify the agreement of the class prediction between the two networks. There is a 100% agreement between the original ReLU network and our SNN, with the fraction of active neurons around 53%. The results of this and the previous paragraph together show that the SNN achieves the same accuracy as the corresponding ANN on ImageNet-like datasets using spiking neurons that fire on average only for 53% of the inputs.

A similar statement is true for the CIFAR100 dataset. Using the same network architecture as for CIFAR10, and pretrained weights downloaded from an online repository [Geifman (2018)], we find on CIFAR100 a 100% agreement between SNN and ReLU predictions with the accuracy of 70.48% and around 38% of active neurons. Thus, on all tested large-scale datasets we find 100 percent agreement between the ANN and SNN indicating that the mapping is without any performance loss.

Sensitivity to noise and parameter changes

As outlined in the introduction, the hardest problem of the conversion is to prevent spike firing in layer n before all spikes from layer $n-1$ have arrived. In our mapping algorithm, a positive value $\zeta > 0$ should guarantee, for a large enough and representative subsample of input images from the training set, that during test the above problem is avoided. For all implementation results so far, the standard choice was $\zeta = 0.5$. In order to check sensitivity to the choice of ζ, we varied ζ across positive and negative values. Using the VGG16 model and the CIFAR10 dataset, we found that the performance degrades gracefully when pushing ζ slightly into the negative regime, but breaks down for a value $\zeta < -0.5$, see Fig. 2.5a. Importantly, when switching from $\zeta = +0.5$ to $\zeta = -0.5$, the total processing time for image classification is reduced by a factor of three.

Noise in hardware implementations could potentially arise from a spike jitter caused for example by imprecisions in detecting the exact time of threshold crossing. We add a Gaussian noise of given standard deviation (SD) and perform 16 trials. No performance degradation is observed up to a standard deviation of 0.001, see Fig. 2.5b. With a jitter of about 1 percent, the accuracy drops from 93.59% to 92.93%, which depending on the application may or may not be considered as acceptable. We note that spike times of hundreds of neurons in a given hidden layer spread over an interval of one or a few time units so that even with a jitter of 0.01 the order of spike firing is considerably changed.

Imprecisions could also arise from heterogeneities in the hardware. A sensitive parameter is the reference slope $\alpha^{(n)}$. We modify the slope parameter in a neuron-specific way $\alpha_i^{(n)} + Y$ where Y is a zero-mean Gaussian random variable with a standard deviation that we control. This simulates a systematic neuron-specific hardware manufacturing imperfection. Even a standard deviation of 0.001 leads to a dramatic drop in performance, see Fig. 2.5c. This is expected since a small mismatch in slope leads to a relatively large shift in spike timing because changes are accumulated throughout the integration interval $[t_{\text{min}}^{(n-1)}, t_{\text{max}}^{(n)}]$. As mentioned in the discussion, using existing learning rules for spiking neurons in the hardware loop [Göltz et al. (2021)] could be used to rapidly fine-tuning weights to compensate for hardware heterogeneities.

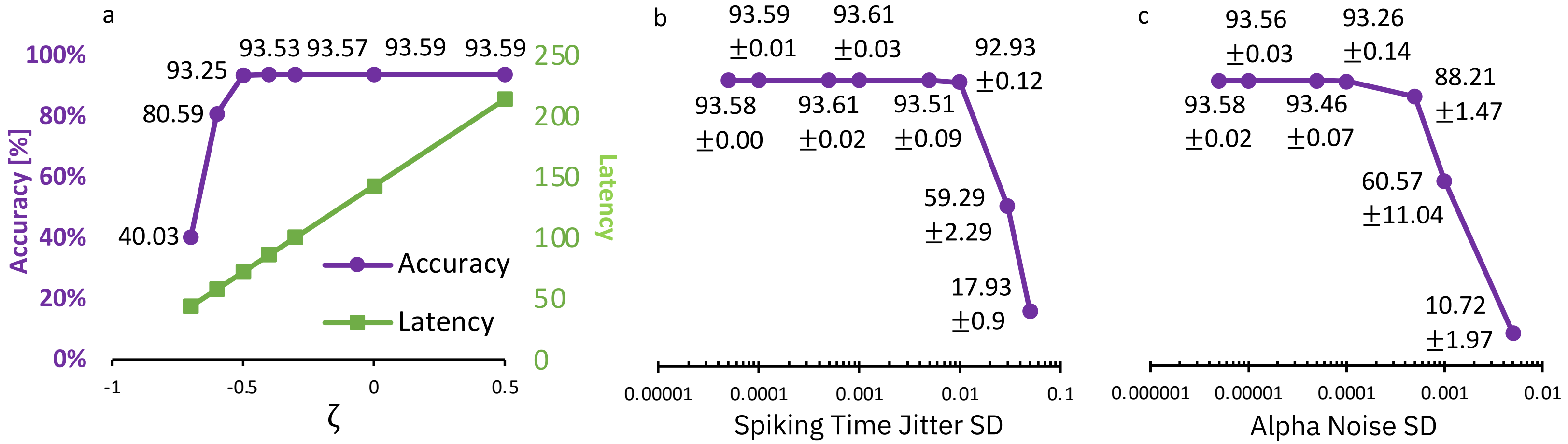


Figure 2.5: **Sensitivity tests a.** Performance as a function of the parameter ζ. Note that the theorem requires $\zeta > 0$; our standard choice in all other figures is $\zeta = 0.5$. **b.** Performance of the network when there is a jitter in spike timing; note that spike timing difference between the earliest and latest neuron in a given layer is often less than 2 time units (see Fig. 2.4). **c.** Performance of the networks when neuron-specific variations are added to the slope parameter $\alpha_i^{(n)}$ during inference; note that the reference value is $\alpha^{(n)} = 1$ for all layers.

2.3 Discussion

In this paper we propose an exact mapping from a ReLU network to an SNN with time-to-first-spike coding. While a relation between ReLU networks and networks of non-leaky integrate-and-fire neurons has been suggested before [Rueckauer and Liu (2018); Zhang et al. (2021); Kheradpisheh and Masquelier (2020); Mirsadeghi et al. (2021)], there have been four obstacles that in the past prevented a successful exact mapping from deep artificial neural networks to deep spiking neural networks:

(i) As mentioned in the introduction, a neuron in layer n that fires a spike before the last spike of neurons in the previous layer $n-1$ has arrived could compromise an exact mapping, since not all inputs are taken correctly into account: in particular, a late inhibitory input could have led to substantially different spiking time if taken into account. Having access to a representative sample of inputs from the training data enables us to solve this problem by an appropriate choice of intervals $[t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}]$, with the condition $t_{\text{max}}^{(n-1)} = t_{\text{min}}^{(n)}$. In other words, firing times of all neurons in layer n are guaranteed to fall into a desired interval, such that all spikes from layer $n-1$ have arrived before the first neuron in layer n fires a spike.

(ii) In some implementations of an SNN, the slope of the potential of a neuron might be negative, zero, or only marginally positive once all input spikes have arrived. In the last case, the threshold could be eventually reached but spiking would be sensitive to noise. We have solved this problem by a positive slope parameter $\alpha_i^{(n)}$ for the trajectory of the integrate-and-fire neuron in combination with a suitable (non-unique) preprocessing of ReLU parameters that together guarantee that the slope of the trajectory is larger than some minimal value once all input spikes have arrived.

(iii) In the past it has been left open how to map the neuron of ReLU that is *inactive* for a given input vector to the corresponding spiking neuron. We have solved this problem by forcing the corresponding spiking neuron to fire a spike at the maximum spike time for that layer. We have also proposed an alternative implementation where inactive neurons do not fire spikes.

(iv) Existing spiking neural network approaches often use rate coding, custom activation functions or specific constraints during ANN training [Rueckauer and Liu (2018); Neftci et al. (2019); Zenke and Ganguli (2018); Zenke and Vogels (2021); Bellec et al. (2020); Woźniak et al. (2020); Yan et al. (2021); Rueckauer et al. (2017); Zhang et al. (2019b); Huh and Sejnowski (2018); Gardner et al. (2015)]. In contrast to prior work, our approach uses sparse TTFS coding, standard ANN elements and does not involve learning. The advantage of our approach in view of an application in neuromorphic edge devices is that a network consisting of standard fully connected and convolutional layers with ReLU activation function as well as max pooling and batch normalization can be pretrained using well-established optimization tools. After conversion, the SNN is guaranteed to have the exact same accuracy as the original ANN. The disadvantage is that hardware imperfections such as uncontrolled parameter variations are not taken into account during training.

TTFS coding for a conversion from ANN to SNN has been used before in an implementation [Rueckauer and Liu (2018)] that contains elements similar to our approach, but with a few important differences. First, we have a systematic way to define the end $t^{(n)}_{\text{max}}$ of the allowed spiking interval. Second, we use a TTFS code with a *linear* relation between spike times and ReLU output whereas the relation is nonlinear in the earlier scheme [Rueckauer and Liu (2018)]. Third, we identify for the case $\bar{w}^{(n)}_{ij} = J^{(n)}_{ij}$ an exact condition for the slope parameter and generalize to mappings where the weights are not simply copied from the ANN to the SNN. The latter gives the freedom to choose the slope parameter so that the trajectory has *always* positive slope.

The success of our method paves the road to many future research direction including both theory and application:

(i) The discrete transition between spikes that are absent or present (depending on the input or on parameter variations) has plagued learning algorithms for spiking neural networks [Göltz et al. (2020); Kheradpisheh and Masquelier (2020); Neftci et al. (2019); Zenke and Ganguli (2018); Bohte et al. (2002); Tavanaei et al. (2019)]. Our theoretical contributions imply that spikes do not appear or disappear, but are rather shifted forward or backward within some finite interval. Earlier learning approaches have shown that those spikes that are triggered at moments when the slope of the potential is close to zero induce a high sensitivity of spike timing to small parameter changes. By introducing a positive slope parameter into an integrate-and-fire neuron in combination with a suitable (non-unique) preprocessing of ReLU parameters our mapping guarantees that the slope of the trajectory is at the moment of firing bounded within some favourable range, so that the problems of sensitivity or discrete transitions are avoided. Therefore, our mapping approach opens the path towards stable learning algorithms in single-spike deep SNNs, for example using the mapping suggested here as initialization of parameters [Stanojevic et al. (2023b)]. Making a step towards increased biological plausibility the approach can also be extended to a leaky integrate-and-fire neuron model where each input spike causes a response described by a double-exponential filter as long as the coding interval $[t^{(n)}_{\text{min}}, t^{(n)}_{\text{max}}]$ is short compared to the two time constants of the exponential [Stanojevic et al. (2023b)].

(ii) Extension of the mapping to other architectures such as ResNet and to other types of neurons beyond non-leaky integrate-and-fire and ReLU would give the opportunity to have higher flexibility in terms of pretrained models. Moreover, it is of interest to further expand the theoretical framework such that it processes not just a single image but a stream of input data. This would present significant benefits for applications. At the moment, our method is limited to feedforward networks and an extension to Recurrent Neural Networks is left for future work.

(iii) To leverage our theoretical contribution for low-energy applications, a hardware implementation of this algorithm is desirable [Widmer et al. (2023)]. In view of future hardware implementations we are interested to further reduce the number of spikes and latency. With an improved implementation in combination with L1 regularization of the ANN, we have already reduced the fraction of spikes per neuron to well below 50%, see Table 2.1. The overall classification latency is defined as $t_{\max}^{(N)}$, which is the time instant when the readout at the output layer happens. The obtained values for $t_{\max}^{(N)}$ in VGG16 networks are around 3000 for MNIST and 2000 for Fashion-MNIST. The shallower networks have much smaller classification latency on the MNIST dataset of around 50 with LeNet5 and close to 15 for a 2-layer fully connected network. Using the VGG16 model, CIFAR10 is classified with a latency close to 200 and around 300 for CIFAR100. Larger datasets yield higher latency, with values around 50’000 for Places365 and close to 100’000 for PASS. We emphasize that the units are arbitrary. The classification latency can be reduced by a less conservative choice of meta-parameters of the mapping so as to reduce the dead time between spike arrival times in layers $n-1$ and layer n. In particular a choice $\zeta = -0.5$ (instead of $\zeta = +0.5$) reduces the overall processing time by a factor of three without a dramatic loss in performance; see Fig. 2.5a. A further reduction of latency is achievable with $\zeta < -0.5$ in combination with retraining of weights and thresholds along the lines discussed above under point (i) [Stanojevic et al. (2023b)].

(iv) For hardware implementations the question of robustness to noise and heterogeneities is also important. We have started to explore the robustness of our algorithm to noise by adding a Gaussian noise of given standard deviation to the spike times of each layer, see Fig. 2.5b. Moreover, we have considered the case where the conversion was done with slope parameter $\alpha^{(n)} = 1$, while the hardware introduces fixed noise of given standard deviation for the slope parameter of each neuron, see Fig. 2.5c. It would be possible to fine-tune network weights with existing algorithm [Göltz et al. (2020)] to compensate for hardware heterogeneities. Our current approach assumes asynchronous processing in continuous time and real-valued weights. In view of digital hardware implementations, it would also be of interest to study the effects of weight and time quantization. Future work on quantification of energy reduction will crucially depend on the concrete hardware implementation that is envisaged.

To summarize, this paper provides a constructive proof that deep ReLU networks and single-spike neural networks of integrate-and-fire neurons are mathematically equivalent. As a consequence, we reach functional deep spiking neural networks that have the same accuracy as ReLU networks and where spiking neurons fire at most one spike per neuron. Since spike transmission is an energy costly process in biology [Attwell and Laughlin (2001)] and neuromorphic hardware [Sorbaro et al. (2020)], our mathematical results open a pathway to low-energy computing with deep neural networks.

2.4 Methods

2.4.1 Preprocessing

Before we perform the mapping from the ReLU network to the SNN, we perform a few preprocessing steps on the network with pretrained weights.

(i) If the network doesn't use batch normalization, this step is skipped. If batch normalization is implemented, it is fused into the neighbouring fully connected and convolutional layers. The parameters of the batch normalization are $\hat{\mu_i}^{(n)}$ and $(\hat{\sigma_i}^{(n)})^2$ denoting the estimated mean and variance, $\gamma_i^{(n)}$ and $\beta_i^{(n)}$ which indicate scaling and shift factors learned during the optimization whereas ϵ is a small constant. In the following equations we use $\kappa_i^{(n)}$ to denote the scaling factor $\gamma_i^{(n)}/\sqrt{(\hat{\sigma_i}^{(n)})^2+\epsilon}$.

When batch normalization is applied to the activation variable $a_i^{(n)}$ and before the activation function, it is fused with the processing of the previous layer (see Fig. 2.2a). The parameters are transformed as follows:

$$b_i^{(n)} \leftarrow \kappa_i^{(n)}(b_i^{(n)} - \hat{\mu_i}^{(n)}) + \beta_i^{(n)} \tag{2.7}$$

$$w_{ij}^{(n)} \leftarrow \kappa_i^{(n)} w_{ij}^{(n)}. \tag{2.8}$$

Note that in case of convolutional architecture each index i corresponds to a different channel.

When batch normalization is applied to the output of the activation function $x_i^{(n)}$, then it is fused with the processing of the subsequent layer (see Fig. 2.2d). The parameters are transformed as follows:

$$b_k^{(n+1)} \leftarrow b_k^{(n+1)} + \sum_i (\beta_i^{(n)} - \kappa_i^{(n)}\hat{\mu_i}^{(n)}) w_{ki}^{(n+1)}, \tag{2.9}$$

$$w_{ki}^{(n+1)} \leftarrow \kappa_i^{(n)} w_{ki}^{(n+1)}. \tag{2.10}$$

Note that the assignments of biases and weights need to be executed in this particular order. Moreover, in case of convolutional architecture, there are a few special cases that need to be considered.

When batch normalization is applied to a zero-padded input into a convolutional layer, the bias change in Eq. (2.9) introduces an unnecessary offset at zero-padded locations. For these particular locations, we calculate the bias by taking into account only the set of inputs S_l which were not obtained through padding (see Fig. 2.2b). Eq. (2.9) is replaced with:

$$b_{k,l}^{(n+1)} \leftarrow b_{k,l}^{(n+1)} + \sum_{i \in S_l} (\beta_i^{(n)} - \kappa_i^{(n)}\hat{\mu_i}^{(n)}) w_{ki}^{(n+1)} \tag{2.11}$$

When max pooling is applied after batch normalization, the weights of the subsequent convolutional layer are changed as described in Eqs. (2.9) and (2.10). The batch normalization multiplies the output of each channel with factor $\kappa_i^{(n)}$, see Eq. (2.10). When this value is negative, the sign of the output is changed. During the inference time, the max pooling operation is

Algorithm 1 Preprocessing

```
    Input: Model with parameters $w_{ij}^{(n)}$ and $b_i^{(n)}$
    Output: $\overline{\text{Model}}$ with parameters $\bar{w}_{ij}^{(n)}$ and $\bar{b}_i^{(n)}$
$x_i^{(0)} \leftarrow \frac{x_i^{(0)}-p}{q-p}$        ▷ Rescale inputs to [0, 1]
for layer ∈ Model do        ▷ Iterate over all layers in the Model
    if layer ∈ [ $\text{Conv}_1$, $\text{FC}_1$ ] and [p, q] ≠ [0,1] then
        Model ← fuse_BN_after_ReLU (Model, p, q)        ▷ Fuse imaginary batch normalization layer in case network was trained on [p, q] range
    else if layer = BN and layer+1 = ReLU then
        Model ← fuse_BN_before_ReLU (Model)        ▷ Fuse batch normalization before activation with previous parametrized layer
    else if layer = BN and layer-1 = ReLU then
        Model ← fuse_BN_after_ReLU (Model)        ▷ Fuse batch normalization after activation with next parametrized layer and process padding or max pooling, Figs. 2.2b, 2.2c
    end if
end for
$c_i^{(0)} \leftarrow 0$, n ← 1
for layer ∈ Model do        ▷ Iterate over all layers in the Model
    if layer ∈ [ $\text{Conv}_\text{n}$, $\text{FC}_\text{n}$ ] then
        $\overline{\text{Model}}$, $c_i^{(n)} \leftarrow$ scale (Model, $c_i^{(n-1)}$, $B_\text{low}$, $\delta$)        ▷ Scale layer with $c_i^{(n-1)}$, Eqs. (2.17), (2.19), and then with $c_i^{(n)}$, Eqs. (2.16), (2.18)
        $X^{(n)} \leftarrow$ max_output ($\mu$, $\overline{\text{Model}}$)        ▷ Calculate maximum output for layer n given training samples $\mu$
        n ← n + 1
    end if
end for
```

transformed into a min pooling operation for the channels with switched sign (see Fig. 2.2c).

(ii) If network has input in range $[0,1]$, this step is skipped. Let's assume that the network has input in arbitrary range $[p,q]$. We would like for the network to operate for input in $[0,1]$ interval without changing its output. This scaling can be seen as an imaginary batch normalization layer between the input layer and the first layer.

The input data is transformed as $x_i^{(0)} \leftarrow \frac{x_i^{(0)}-p}{q-p}$ and the biases and weights of the first layer are set to:

$$b_k^{(1)} \leftarrow b_k^{(1)} + p\sum_i w_{ki}^{(1)}, \tag{2.12}$$

$$w_{ki}^{(1)} \leftarrow (q-p)\, w_{ki}^{(1)}. \tag{2.13}$$

When there is zero padding in the first convolutional layer, Eq. (2.12) is replaced with:

$$b_{k,l}^{(1)} \leftarrow b_{k,l}^{(1)} + p \sum_{i \in S_l} w_{ki}^{(1)} \tag{2.14}$$

(iii) In order to guarantee that the potential increases once all input spikes have arrived, we rescale the parameters of the ReLU network. We exploit the scaling symmetry of ReLU neurons $[a_i]_+ = C[a_i/C]_+$, for $C > 0$ and normalize weights so that the sum of input weights is smaller than $1-\delta$, for some $0 < \delta < 1$. Similarly, we want to make sure that the sum of input weights does not fall below some lower bound $(-B_{\text{low}}) < 0$. To implement the scaling, we begin from the initial weights $\bar{w}_{ij}^{(n)} \leftarrow w_{ij}^{(n)}$ and biases $\bar{b}_i^{(n)} \leftarrow b_i^{(n)}$, start in layer $n = 1$ and proceed up to $n = M$ one layer at a time. For each neuron i, we calculate the sum over all the incoming weights

$$c_i^{(n)} = \sum_j \bar{w}_{ij}^{(n)} \tag{2.15}$$

If $c_i^{(n)} > (1-\delta)$, we set for this specific neuron i its bias and incoming weights (for all j) to

$$\bar{b}_i^{(n)} \leftarrow \frac{(1-\delta)}{c_i^{(n)}} \bar{b}_i^{(n)}; \quad \text{and} \quad \bar{w}_{ij}^{(n)} \leftarrow \frac{(1-\delta)}{c_i^{(n)}} \bar{w}_{ij}^{(n)} \tag{2.16}$$

and the outgoing weights (for all k) to

$$\bar{w}_{ki}^{(n+1)} \leftarrow \frac{c_i^{(n)}}{1-\delta} \bar{w}_{ki}^{(n+1)} \tag{2.17}$$

Similarly, if $c_i^{(n)} \leq -B_{\text{low}}$ we set the bias and the incoming weights (for all j) to

$$\bar{b}_i^{(n)} \leftarrow \frac{B_{\text{low}}}{|c_i^{(n)}|} \bar{b}_i^{(n)}; \quad \text{and} \quad \bar{w}_{ij}^{(n)} \leftarrow \frac{B_{\text{low}}}{|c_i^{(n)}|} \bar{w}_{ij}^{(n)} \tag{2.18}$$

and the outgoing weights (for all k) to

$$\bar{w}_{ki}^{(n+1)} \leftarrow \frac{|c_i^{(n)}|}{B_{\text{low}}} \bar{w}_{ki}^{(n+1)} \tag{2.19}$$

Note that signs are not changed by the scaling operation. Scaling ensures that for all hidden layers $(-B_{\text{low}}) \leq \sum_j \bar{w}_{ij}^{(n)} < 1$. We have larger weights in the final output layer (readout weights), but this does not cause any problems. The network where all the above preprocessing steps are applied is called a *scaled ReLU network*. Its parameters are denoted with a bar to distinguish them from the original, unscaled, network.

If $c_i^{(n)} > (1-\delta)$ the activation variable $\bar{a}_i^{(n)}$ is given by

$$\bar{a}_i^{(n)} \leftarrow \frac{(1-\delta)}{c_i^{(n)}} a_i^{(n)} \tag{2.20}$$

and if $c_i^{(n)} \le -B_{\mathrm{low}}$ by

$$\bar{a}_i^{(n)} \leftarrow \frac{B_{\mathrm{low}}}{|c_i^{(n)}|} a_i^{(n)} \tag{2.21}$$

and $\bar{a}_i^{(n)} \leftarrow a_i^{(n)}$ otherwise. Similarly, if $c_i^{(n)} > (1-\delta)$ the output $\bar{x}_i^{(n)}$ of ReLU is given by

$$\bar{x}_i^{(n)} \leftarrow \frac{(1-\delta)}{c_i^{(n)}} x_i^{(n)} \tag{2.22}$$

and if $c_i^{(n)} \le -B_{\mathrm{low}}$ as

$$\bar{x}_i^{(n)} \leftarrow \frac{B_{\mathrm{low}}}{|c_i^{(n)}|} x_i^{(n)} \tag{2.23}$$

and $\bar{x}_i^{(n)} \leftarrow x_i^{(n)}$ otherwise.

(iv) We apply all training data $1 \le \mu \le P$ at the input layer of the scaled ReLU network and observe the activation pattern for each neuron in the network. For each layer n we determine the maximal output of the activation function $\bar{x}_i^{(n)}(\mu)$ across all training data $1 \le \mu \le P$ and all neurons i in that layer:

$$X^{(n)} = \max_{i,\mu}\{\bar{x}_i^{(n)}(\mu)\} \tag{2.24}$$

If the number P is very large, we choose a statistically representative subset of data and perform the max-operation over these.

2.4.2 Conversion to SNN

The essential idea of the mapping from the ReLU neurons to the spiking neurons is that a positive activation leading to an output $\bar{x}_i^{(n)} = \bar{a}_i^{(n)} > 0$ is identified with an *early* firing time: $t_i^{(n)} = t_{\max}^{(n)} - \bar{a}_i^{(n)}$, whereas vanishing output $\bar{x}_i^{(n)} = 0$ corresponds to firing at $t_i^{(n)} = t_{\max}^{(n)}$.

The actual mapping is defined as follows (see Fig. 2.3).

(i) Input encoding. The input data lies in the interval $0 \le x_i^{(0)} < 1$ and we set $t_{\min}^{(0)} = 0$, $t_{\max}^{(0)} = 1$ and $t_i^{(0)} = 1 - x_i^{(0)}$. With the parameters of the input layer fixed, we now proceed layer by layer from $n = 1$ to $n = M$

(ii) We set $t_{\min}^{(n)} = t_{\max}^{(n-1)}$

(iii) We set $t_{\max}^{(n)} = t_{\min}^{(n)} + B^{(n)}$ with $B^{(n)} = (1+\zeta)\,X^{(n)}$ and $\zeta > 0$. The idea is that even the neuron with the strongest input must fire within the desired interval $[t_{\min}^{(n)}, t_{\max}^{(n)}]$, i.e., not too early. Under the assumption that the test data comes from the same statistical distribution as the

Algorithm 2 Conversion

```
Input: $\overline{\text{Model}}$ with parameters $\bar{w}_{ij}^{(n)}$ and $\bar{b}_i^{(n)}$, $X^{(n)}$
Output: SpModel with parameters $J_{ij}^{(n)}$, $\vartheta_i^{(n)}$ $\alpha^{(n)}$, $t_{\text{min}}^{(n)}$ and $t_{\text{max}}^{(n)}$
$t_{min}^{(0)} \leftarrow 0,\ t_{max}^{(0)} \leftarrow 1,\ n \leftarrow 1,\ \alpha^{(n)} \leftarrow 1, \forall n \leftarrow 1..M$
for layer $\in \overline{\text{Model}}$ do        ▷ Iterate layer-wise from the first to the output layer and calculate parameters and intervals
    if layer $\in$ [ $\overline{\text{ConvBN}_\text{n}}$, $\overline{\text{FCBN}_\text{n}}$ ] then
        if layer+1=ReLU then
            $t_{min}^{(n)} \leftarrow t_{max}^{(n-1)}$
            $t_{max}^{(n)} \leftarrow t_{max}^{(n-1)} + (1+\zeta) X^{(n)}$
            $J_{ij}^{(n)} \leftarrow \frac{\alpha^{(n)} \bar{w}_{ij}^{(n)}}{1 - \sum \bar{w}_{ij}^{(n)}}$
            $\vartheta_i^{(n)} \leftarrow \alpha^{(n)}(t_{max}^{(n)} - t_{min}^{(n-1)}) + \sum_i J_{ij}^{(n)}(t_{max}^{(n)} - t_{min}^{(n)}) - (\alpha^{(n)} + \sum_i J_{ij}^{(n)}) \bar{b}_i^{(n)}$
        else if layer+1=softmax then
            $\alpha_i^{(n)} \leftarrow \frac{\bar{b}_i^{(n)}}{(t_{max}^{(n-1)} - t_{min}^{(n-1)})}$
            $J_{ij}^{(n)} \leftarrow \bar{w}_{ij}^{(n)}$
        end if
        $n \leftarrow n+1$
    end if
end for
```

training data, a small value $\zeta \ll 1$ should in practice provide a sufficient safety margin. Indeed, if the training data set is large enough to be statistically representative, the probability that test data contains a point causing activation larger than $(1+\zeta)\, X^{(n)}$ decreases rapidly with ζ. The range $[t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}]$ is therefore large enough to encode all the values from layer n of the rescaled ReLU network.

(iv) For a given $\alpha_i^{(n)} > 0$ we first choose a reference threshold $\tilde{\vartheta}_i^{(n)}$ in layer n such that an integrator without any spike input would fire at $t_{\text{max}}^{(n)}$. Hence for $t > t_{\text{min}}^{(n)}$ the reference threshold is

$$\tilde{\vartheta}_i^{(n)} = \alpha_i^{(n)}\, [t_{\text{max}}^{(n)} - t_{\text{min}}^{(n-1)}] \tag{2.25}$$

For the formal proof of the exact mapping, we set the reference threshold for $t \le t_{\text{min}}^{(n)}$ to a sufficiently high value $\tilde{\vartheta}_i^{(n)} = \Theta \to \infty$ for all times $t \le t_{\text{min}}^{(n)}$. This ensures that no neuron in layer n fires before $t_{\text{min}}^{(n)}$. The value from Eq. (2.25) is used only for $t > t_{\text{min}}^{(n)}$. However, for our practical algorithmic implementations we use the threshold given in Eq. (2.25) throughout for all t, because in all encountered data sets the probability of neurons in layer n firing before $t = t_{\text{min}}^{(n)}$ was negligible.

(v) The actual threshold also depends on the bias and weights of the neuron. To account for this, we set the actual threshold of neuron i in layer n to a value

$$\vartheta_i^{(n)} = \tilde{\vartheta}_i^{(n)} + D_i^{(n)} \tag{2.26}$$

With these parameter choices, an exact mapping from ReLU network to an SNN is possible with a value

$$D_i^{(n)} = [B^{(n)} \sum_j J_{ij}^{(n)}] - [\alpha_i^{(n)} + \sum_j J_{ij}^{(n)}]\, \bar{b}_i^{(n)}. \tag{2.27}$$

and weights

$$J_{ij}^{(n)} = \frac{\alpha_i^{(n)}}{1 - \sum_{j'} \bar{w}_{ij'}^{(n)}}\, \bar{w}_{ij}^{(n)} \tag{2.28}$$

where $\bar{w}_{ij}^{(n)}$ are the weights of the scaled ReLU network. Note that the denominator of Eq. (2.28) is always positive since $\sum_j \bar{w}_{ij}^{(n)} < 1$. Hence the mapping does not change the sign of the weights. The inverse weight transform from SNN to ReLU is

$$\bar{w}_{ij}^{(n)} = \frac{1}{\alpha_i^{(n)} + \sum_{j'} J_{ij'}^{(n)}} J_{ij}^{(n)} \tag{2.29}$$

This completes the conversion.

Note that we kept biases $\bar{b}_i^{(n)}$ as explicit parameters. However, following standard practice in the ANN literature, we could replace biases by an additional input neuron with connection weight equal to $\bar{b}_i^{(n)}$. The equations above as well as those for weight rescaling in Phase 1 are to be used analogously in that case.

Lemma. With the conversion rules Eqs. (2.25) to (2.28) spike firing occurs at a value $\bar{x}_i^{(n)} = t_{\max}^{(n)} - t_i^{(n)}$.

Proof. Let us integrate the differential equation (2.2) of the integrate-and-fire units which yields for $t_{\min}^{(n)} < t < t_{\max}^{(n)}$ a voltage

$$V_i^{(n)}(t) = [t - t_{\min}^{(n-1)}]\, \alpha_i^{(n)} + \sum_j J_{ij}^{(n)} [t - t_j^{(n-1)}], \tag{2.30}$$

where all neurons in layer $n-1$ have firing times $t_j^{(n-1)} \leq t_{\max}^{(n-1)} = t_{\min}^{(n)}$. The firing time $t_i^{(n)}$ of neuron i in layer n is given by the threshold condition $V_i^{(n)}(t_i^{(n)}) = \vartheta_i^{(n)}$. We exploit that neurons in layer $n-1$ that have not yet fired are forced to fire at $t_{\max}^{(n-1)}$. We now insert the claims $t_i^{(n)} = t_{\max}^{(n)} - \bar{x}_i^{(n)}$ and $t_j^{(n-1)} = t_{\max}^{(n-1)} - \bar{x}_j^{(n-1)}$ into Eq. (2.30) and use Eqs. (2.25), (2.26),(2.27) as well as $t_{\min}^{(n)} = t_{\max}^{(n-1)}$ and $B^{(n)} = t_{\max}^{(n)} - t_{\min}^{(n)}$ to find

$$\bar{x}_i^{(n)} = \frac{1}{\alpha_i^{(n)} + \sum_j J_{ij}^{(n)}} \sum_j J_{ij}^{(n)} \bar{x}_j^{(n-1)} + b_i^{(n)} \tag{2.31}$$

Thus, the Eq. (2.29) for the weights follows from a comparison of this formula with the ReLU equation $\bar{x}_i^{(n)} = \sum_j \bar{w}_{ij}\bar{x}_j^{(n-1)}$; see Eq. (2.1) with $\bar{x}_i^{(n)} = [\bar{a}_i^{(n)}]_+$. The solution is unique since trajectories have positive slope so that the threshold is reached at most once.

2.4.3 Conversion of Max pooling

If the ReLU network contains max pooling layers, the SNN contains layers performing max pooling and min pooling, outputting the earliest and latest spiking time respectively. This functionality can be implemented with integrate-and-fire neurons such that each neuron fires exactly one spike. To this end we introduce connections $K_{ij}^{(n-1)}$ within a given layer. A spike at time $t_j^{(n-1)}$ of a ReLU neuron j in layer $n-1$ generates a pulse current, modeled by a Dirac delta pulse of total charge $K_{ij}^{(n-1)}$, which is injected into neuron i of the max pooling or min pooling operation belonging to the layer $n-1$. The voltage of neuron i evolves according to

$$\frac{\mathrm{d}V_{i(\mathrm{MMP})}^{(n-1)}}{\mathrm{d}t} = \sum_j K_{ij}^{(n-1)}\delta(t - t_j^{(n-1)}) \tag{2.32}$$

If $V_{i(\mathrm{MMP})}^{(n-1)}$ crosses the threshold $\vartheta_{i(\mathrm{MMP})}^{(n-1)}$ at time t then $t = t_{i(\mathrm{MMP})}^{(n-1)}$ is the firing time of neuron i. For the layers which are preceded by a max pooling or min pooling operation the Eq. (2.2) is replaced with:

$$\frac{\mathrm{d}V_i^{(n)}}{\mathrm{d}t} = \alpha_i^{(n)}\, H(t - t_{\mathrm{min}}^{(n-1)}) + \sum_j J_{ij}^{(n)} H(t - t_{j(\mathrm{MMP})}^{(n-1)}) + I_i^{(n)}(t) \tag{2.33}$$

In case of the max pooling operation, all weights $K_{ij}^{(n)}$ are set to slightly larger values than the threshold value $\vartheta_{i(\mathrm{MMP})}^{(n)}$, such that the very first input spike triggers firing. In case of min pooling operation, parameters $K_{ij}^{(n)}$ are set to the value of $\vartheta_{i(\mathrm{MMP})}^{(n)}/Q < K_{ij}^{(n)} < \vartheta_{i(\mathrm{MMP})}^{(n)}/(Q-1)$ where Q is the total number of inputs. As a consequence, the very last input spike triggers the firing.

2.4.4 Mapping of the output layer

The output layer of the scaled ReLU network has a softmax activation function and parameters $\{\bar{w}_{ij}^{(M+1)}, \bar{b}_i^{(M+1)}\}$. In the SNN we implement the output layer with an integrator unit, i.e. the neurons just integrate the currents and do not spike spike. A spike arriving at the output layer at time $t_j^{(M)}$ from a neuron in layer M generates a step current input with amplitude $\bar{w}_{ij}^{(M+1)}$ into neuron i of layer $M+1$. The voltage of neuron i in layer $M+1$ evolves according to

$$\frac{\mathrm{d}V_i^{(M+1)}}{\mathrm{d}t} = \alpha_i^{(M+1)}\, H(t - t_{\mathrm{min}}^{(M)}) + \sum_j \bar{w}_{ij}^{(M+1)} H(t - t_j^{(M)}) \tag{2.34}$$

where H denotes the Heaviside step function. The non-leaky integration starts at time $t_{\min}^{(M)}$ and lasts until time $t_{\max}^{(M)}$ and $\alpha_i^{(M+1)}$ takes value:

$$\alpha_i^{(M+1)} = \frac{\bar{b}_i^{(M+1)}}{t_{\max}^{(M)} - t_{\min}^{(M)}} \tag{2.35}$$

The largest potential $V_i^{(M+1)}$ at time $t_{\max}^{(M)}$ determines the prediction.

2.4.5 Final remarks regarding the mapping

First, as mentioned in the results section, other mappings are also possible. For efficient coding with short latency, the aim is to choose parameters such that the resulting time intervals $[t_{\min}^{(n)}, t_{\max}^{(n)}]$ are not too large, however large enough to encode all values from the ReLU network with sufficient temporal resolution and such that the firing times of different layers do not overlap. Note that (in contrast to leaky integration with time-constant τ) a non-leaky integrator has no intrinsic time scale. Second, it would be possible to start the integration of all integrate-and-fire units across all layers n synchronously at time $t = 0$, if we increase at the same time the threshold in layer n by an amount $\alpha^{(n)}\, t_{\min}^{(n)}$.

Third, since $\alpha_i^{(n)} + \sum_j J_{ij}^{(n)} > 0$ and all neurons in layer $n-1$ have fired at or before $t_{\max}^{(n-1)}$, the voltage trajectories $V_i^{(n)}$ of all neurons i in layer n have for $t > t_{\max}^{(n-1)} = t_{\min}^{(n)}$ a positive slope; see. Eq. (2.2). If, after preprocessing, $\sum_j \bar{w}_{ij}^{(n)} \leq 1 - \delta$, then the maximal slope of the trajectory at threshold is $\alpha^{(n)}/\delta$. Similarly, if after preprocessing $\sum_j \bar{w}_{ij}^{(n)} \geq -B_{\text{low}}$, then the minimal slope at the moment of firing is $\alpha^{(n)}/(1 + B_{\text{low}})$. A small slope of the potential close to the threshold should be avoided, since this increases the sensitivity to noise (in particular in view of combining with learning algorithms or unknown heterogeneities in the exact value of the slope). In practice, a value of $B_{\text{low}} = 10$ worked fine for our numerical simulations.

Fourth, we used a value of $\zeta = 0.5$. If the training set is large, and if we have access to all data in the training set, a positive but small $\zeta \to 0$ would be sufficient to guarantee that a neuron cannot fire 'too early'. However, the test set could potentially include data where the total activation is slightly larger than the maximal activation in the training set. Since training set and test set arise, in principle, from the same statistical distribution, a parameter choice $\zeta = 0.5$ should provide a sufficient safety margin and this is confirmed in our simulations in the Results section.

2.4.6 Datasets

All the experiments were performed with Python programming language and TensorFlow library. The simulations were executed on NVIDIA A100 GPUs. We consider six datasets of different sizes and complexity:

(i) MNIST and Fashion MNIST datasets contain greyscale images of size 28 × 28 which are labeled into ten classes. For each of the two datasets there are 60000 training images and 10000 testing images. Data preprocessing step includes normalizing pixel values to the $[0,1]$ range and in the case of a fully connected network the input is also reshaped. The pretrained parameters of the original ReLU networks are obtained by training with backpropagation using Adam optimizer [Kingma and Ba (2014)] with exponential learning rate schedule and standard cross-entropy loss. We apply dropout for regularization. In case of the VGG16 architecture the kernel was always of size 3 and the input of each convolutional operation is zero padded such that the shape at the output remains the same. Due to small input size, the first max pooling operation in the standard VGG16 architecture is omitted. The output of the convolutional part of VGG16 is of size 512 which is followed by two fully connected layers each containing 512 neurons and the output layer. The LeNet5 architecture has three convolutional, two max pooling and two fully connected layers with 84 and 10 neurons, see Fig 2.2a. Finally, the 2-layer fully connected network has one hidden layer with 600 units. LeNet5 and VGG16-like networks also contain batch normalization before and after ReLU function, respectively.

In Fig. 2.2a we see the scaled LeNet5 network where the batch normalization are fused with previous convolutional and fully connected layers and the parameters of the network are scaled. For VGG16 network the batch normalization are fused with next convolutional and fully connected layers. Moreover, in this case the shift which appears due to zero padding is counter balanced with bias change at certain locations, see Fig. 2.2b, and every time batch normalization appears before max pooling, the channels whose sign is changed are replaced with min pooling, see Fig. 2.2c. Since the model is trained on $[0,1]$ range there is no need to fuse an imaginary batch normalization after the input. In order to obtain the scaled ReLU network the parameters of the network are scaled. Finding the maximum output $X^{(n)}$ of each layer on the subset of the training set finalizes the preprocessing step (see Fig. 2.1). In the following mapping phase the parameters of SNN are calculated.

(ii) CIFAR10 and CIFAR100 contain images of size 32 × 32 × 3 [Krizhevsky et al. (2009)]. For each of the two datasets there are 50000 training images and 10000 testing images. The data preprocessing step includes normalizing data with given fixed mean and standard deviation as given in [Geifman (2018)]. The network was trained on the data rounded to $[-3,3]$ range. In preparation for SNN mapping and inference, the input $x_i^{(0)}$ is further preprocessed as $x_i^{(0)} \leftarrow \frac{x_i^{(0)}+3}{6}$. The kernel is always of size 3 and the input of each convolutional operation is zero padded such that the shape at the output stays the same. The output of the convolutional part of the VGG16 architecture has size 512 which is followed by two fully connected layers with 512 and 10 neurons.

During the preprocessing, the batch normalization is fused with the next convolutional and fully connected layers and bias is changed in locations where the input is coming from the zero padding. When necessary, the max pooling function is replaced with min pooling. Since the model is trained on $[-3,3]$ range the imaginary batch normalization is fused with first convolutional layer and in locations where the input is generated by zero padding the bias is changed. In order to obtain the scaled ReLU network the parameters of the network are scaled. Finding the maximum output $X^{(n)}$ of each layer on the subset of training set finalizes the preprocessing step (see Fig. 2.1). In the following mapping phase the parameters of SNN are calculated.

(iii) The images in Places365-Standard dataset are labeled into 365 scene categories. There are 1.8 million training images, 36500 validation images and 328500 test images. Since the labels for the test set are not publicly available, we report the metrics on the validation set. Data preprocessing step includes centralizing data around a given fixed mean and reshaping it to the size of 224 × 224 × 3 as described in [Zhou (2018)]. The network is trained on the data which can be rounded to $[-200, 200]$ interval. In preparation for SNN mapping and inference the input $x_i^{(0)}$ is further preprocessed as $x_i^{(0)} \leftarrow \frac{x_i^{(0)}+200}{400}$. Since the model is trained on $[-200, 200]$ range the imaginary batch normalization is fused with first convolutional layer and in locations where the input is generated by zero padding the bias is changed. In order to obtain the scaled ReLU network the parameters of the network are scaled. Finding the maximum output $X^{(n)}$ of each layer on the subset of training set finalizes the preprocessing step (see Fig. 2.1). In the following mapping phase the parameters of SNN are calculated.

(iv) We randomly sample 100000 testing and 5000 training images from PASS dataset. Most of the images in the dataset are colored and the few ones that are not are dropped during data preprocessing. The images are reshaped to 224 × 224 × 3 and preprocessed with the same function as ImageNet for VGG16, which includes centering each color channel around zero mean. Since the model is trained on $[-200, 200]$ range, in preparation for SNN mapping and inference, the input $x_i^{(0)}$ is further preprocessed as $x_i^{(0)} \leftarrow \frac{x_i^{(0)}+200}{400}$ situating the input on the $[0, 1]$ range. Moreover, the imaginary batch normalization is fused with the first convolutional layer and in locations where the input is generated by zero padding the bias is changed. In order to obtain the scaled ReLU network the parameters of the network are scaled. Finding the maximum output $X^{(n)}$ of each layer on the subset of training set finalizes the preprocessing step (see Fig. 2.1). In the following mapping phase the parameters of SNN are calculated.

3 Are training trajectories of deep single-spike and deep ReLU network equivalent?

Paper information

Authors: Ana Stanojevic, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini, Angeliki Pantazi, Wulfram Gerstner

Abstract Communication by binary and sparse spikes is a key factor for the energy efficiency of biological brains. However, training deep spiking neural networks (SNNs) with backpropagation is harder than with artificial neural networks (ANNs), which is puzzling given that recent theoretical results provide exact mapping algorithms from ReLU to time-to-first-spike (TTFS) SNNs. Building upon these results, we analyze in theory and in simulation the learning dynamics of TTFS-SNNs. Our analysis highlights that even when an SNN can be mapped exactly to a ReLU network, it cannot always be robustly trained by gradient descent. The reason for that is the emergence of a specific instance of the vanishing-or-exploding gradient problem leading to a bias in the gradient descent trajectory in comparison with the equivalent ANN. After identifying this issue we derive a generic solution for the network initialization and SNN parameterization which guarantees that the SNN can be trained as robustly as its ANN counterpart. Our theoretical findings are illustrated in practice on image classification datasets. Our method achieves the same accuracy as deep ConvNets on CIFAR10 and enables fine-tuning on the much larger PLACES365 dataset without loss of accuracy compared to the ANN. We argue that the combined perspective of conversion and fine-tuning with robust gradient descent in SNN will be decisive to optimize SNNs for hardware implementations needing low latency and resilience to noise and quantization.

Publication The chapter is under review at NeurIPS 2023 conference.

3.1 Introduction

Similar to the brain, neurons in spiking neural networks (SNNs) communicate via short pulses called spikes – in striking contrast to artificial neural networks (ANNs) where neurons communicate by the exchange of real-valued signals. While ANNs are the basis of modern artificial intelligence (AI) with impressive achievements [Brown et al. (2020); Jaegle et al. (2021); Yu et al. (2021)], their high performance on various tasks comes at the expense of high energy consumption [Strubell et al. (2020); Patterson et al. (2022); Wu et al. (2022)]. In general, high energy consumption is a challenge in terms of sustainability or deployment in low-power edge devices [Wang et al. (2020); Boroumand et al. (2021); Jiang et al. (2018)]. Due to their sparse binary communication scheme, SNNs may offer a potential solution by reducing resource usage in the network [Burr et al. (2017); Sebastian et al. (2018); Göltz et al. (2020); Gallego et al. (2020); Göltz et al. (2021); Davies et al. (2021); Diehl et al. (2016)], but these studies have shown that it is difficult to demonstrate working SNNs which perform at the same level as ANNs.

There exist multiple methods to train the parameters of an SNNs with various advantages and drawbacks. Traditionally, SNNs were trained with plasticity rules observed in biology [Masquelier and Thorpe (2007); Kheradpisheh et al. (2018); Illing et al. (2019)] but it appears more efficient to rely on gradient-descent optimization as done in deep learning (see [Markram et al. (2011); Dellaferrera and Kreiman (2022); Scellier and Bengio (2017); Meulemans et al. (2021); Illing et al. (2021); Bellec et al. (2020)] for theoretical relationships between the plasticity rules and gradient descent). One of the most successful training paradigms for SNNs views the spiking neuron as discrete-time recurent unit with binary activation and uses a pseudo-derivative or surrogate gradient on the backward pass while keeping the strict threshold function in the forward pass [Neftci et al. (2019); Bellec et al. (2018); Zenke and Ganguli (2018); Woźniak et al. (2020); Huh and Sejnowski (2018)]. Other approaches [Schmitt et al. (2017); Gardner et al. (2015); Stanojevic et al. (2023a)] either translate ANN activations into SNN spike counts to train the SNN with the ANN gradients, or use temporal coding with a large number of spikes, both of which can jeopardize the energy efficiency of SNNs.

More recently, and in contrast to spike-count measures in neuroscience [Hubel and Wiesel (1959)], it was found in sensory brain areas that neurons also encode information in the exact timing of the first spike, i.e. more salient information leads to earlier spikes [Gollisch and Meister (2008); Johansson and Birznieks (2004); Kubke et al. (2002)] which in turn leads to a fast response to the stimuli [Thorpe et al. (1996, 2001)]. While it is possible to train temporally coded spiking neural networks where neurons send *multiple spikes* to transmit information, we focus in this paper on a *time-to-first-spike* (TTFS) coding scheme [Gerstner (1998)] where each neuron fires at most a single spike. The goal of the present study is (1) to analyze theoretically why all attempts at training SNNs with TTFS encoding run into difficulties with deep networks (beyond 4 layers) and (2) to provide a solution to these problems.

Related work There is a long history of implementations of gradient descent in SNNs with TTFS. In [Bohte et al. (2002)] the authors used the Spike Response Model [Gerstner (1998)] and calculated backpropagation gradients with respect to spike timing and parameters. While the paper states that the learning rule contains an approximation, it turns out to be the exact gradient when the number of spikes is fixed to avoid discontinuities, i.e. spikes do not appear or disappear. This approach was rediscovered recently, extended to other neuron models [Göltz et al. (2021); Wunderlich and Pehle (2021); Zhang et al. (2021); Mostafa (2018); Stanojevic et al. (2022a)] and applied to small machine learning datasets like MNIST and Fashion-MNIST [Göltz et al. (2021); Comşa et al. (2021); Zhang et al. (2021); Stanojevic et al. (2022a)] with architectures of 4 hidden layers or less.

A different line of work avoids training altogether and converts directly an ANN into an SNN. Beyond the classic conversion techniques based on rate coding [Rueckauer et al. (2017); Hunsberger and Eliasmith (2016)], some studies considered the conversion from ANNs to temporally coded SNNs [Stockl and Maass (2021); Bu et al. (2022); Rueckauer and Liu (2018); Stanojevic et al. (2022b)]. While most of them relied on an inexact mapping algorithm or unconventional threshold dynamics, it was recently shown that approximation-free conversion from ANN to TTFS-SNN is possible [Stanojevic et al. (2022b)]. However, these results have not shown any benefits outside of the network conversion setting, therefore it remains difficult to understand why gradient descent in the TTFS cannot be used for training or fine-tuning deep SNNs. We build upon this aforementioned work [Stanojevic et al. (2022b)] to study the learning dynamics of SNNs.

Contributions of the paper Our work combines the exact backpropagation update steps [Wunderlich and Pehle (2021); Zhang et al. (2021); Mostafa (2018); Stanojevic et al. (2022a)] with an exact mapping between ANNs and SNNs [Stanojevic et al. (2022b)], and goes considerably beyond the state-of-the-art with respect to the following points:

- **Even if a TTFS-SNN has an equivalent ReLU ANN, they do not necessarily follow equivalent learning trajectories.** We extend the theory from [Stanojevic et al. (2022b)] to provide a reversed mapping from TTFS-SNN to an equivalent ReLU network. We propose the conditions and an adaptive SNN hyperparameter update rule, which are necessary for the equivalence to hold throughout training. Furthermore, the *linearly mappable condition* from SNN to ANN parameters is identified as the one that guarantees that gradient descent follows the learning trajectories of the equivalent ReLU ANN.

- **Hard instance of the vanishing-gradient problem.** We identify that naively using ANN weight matrix initialization techniques [Bengio et al. (1994); Hochreiter et al. (2001); Goodfellow et al. (2016)] with TTFS-SNN results in a severe instance of the vanishing-gradient problem. We identify the problem analytically and provide a generic recipe to solve it and initialize TTFS-SNN efficiently.

- **Training SNNs to the state-of-the-art accuracy on large datasets.** All previous learning attempts in the TTFS setting were limited to MNIST or Fashion-MNIST datasets and networks of up to 4 layers [Göltz et al. (2021); Comşa et al. (2021); Kheradpisheh and Masquelier (2020); Mostafa (2018); Zhang et al. (2021)]. We are the first to train TTFS-SNN on CIFAR10 [Krizhevsky et al. (2009)], CIFAR100 [Krizhevsky et al. (2009)] and PLACES365 [Zhou et al. (2017)]. As predicted by the theory, on all datasets, our TTFS-SNN achieves the exact same performance as a ReLU network with the same architecture.

- **Demonstrating the benefits of training under hardware constraints.** Since SNNs can be implemented in low-energy neuromorphic hardware, we test their robustness to quantization of spike times and weights, by fine-tuning the quantized network with our training framework. We show that the latency to decision can be reduced by a factor of four with a performance drop of less than 3 percent on CIFAR10.

3.2 Definition and properties of time-to-first-spike networks

In the following section, we will analyze the gradient descent dynamics in a TTFS setting. To avoid any approximation we follow [Stanojevic et al. (2022b)] and study deep spiking neural networks consisting of neurons with triangular post-synaptic integration filters. This model should be viewed as the linearization of a more classical double-exponential filter [Göltz et al. (2021); Comşa et al. (2021)] where all the spikes of a layer arrive within a time window that is small compared to the two time constants of the double-exponential. As we will see in the following section, the theory becomes rigorous under this linearized approximation – we discuss the extension to a spike response model with a double-exponential post-synaptic potential filter in Section 3.6.1.

A time-to-first-spike (TTFS) network model The neurons are arranged in N hidden layers where the spikes of neurons in layer n are sent to neurons in layer $n+1$. The layers are either fully-connected (i.e. each neuron receives input from all neurons in the previous layer) or convolutional (i.e. connections are limited to be local and share weights). All connections are feed-forward, i.e. there are no recurrent connections.

Input layer: At the first layer, the analog input to the network represents, for example, the pixel intensity. We assume that the input is scaled to the interval $[0,1]$ and the values are encoded with TTFS coding (a high input pixel intensity $x_j^{(0)}$ leads to an early spike at $t_j^{(0)}$):

$$t_j^{(0)} = \tau_c[1 - x_j^{(0)}] = t_{\max}^{(0)} - \tau_c x_j^{(0)} \tag{3.1}$$

where spiking time $t_j^{(0)}$ of neuron j in the input layer encodes the real-valued input $x_j^{(0)} \in [0,1]$ and $t_{\max}^{(0)} = \tau_c$ is the last possible spike time in the input layer. The conversion parameter τ_c translates unit-free inputs into time units. In biology τ_c in sensory areas is in the range of a few milliseconds [Gollisch and Meister (2008); Johansson and Birznieks (2004)] whereas in hardware devices it could be in the range of microseconds or even shorter.

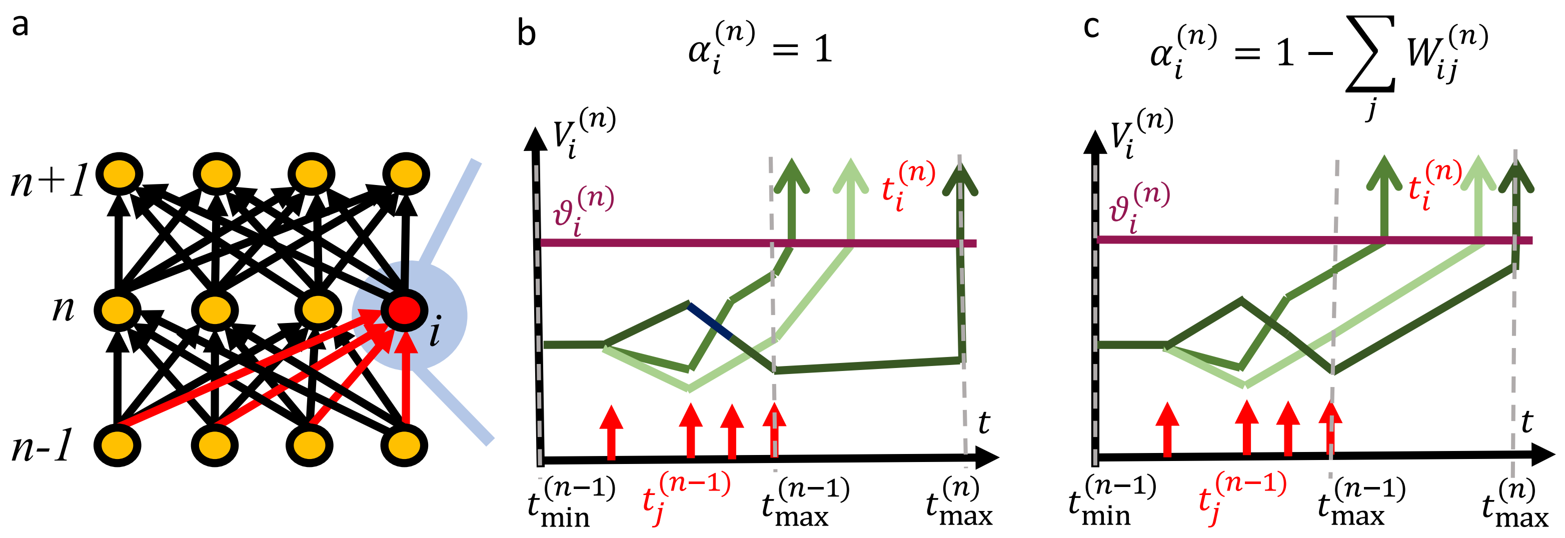


Figure 3.1: **Network of TTFS neurons. a.** A feed-forward TTFS architecture. **b.** Potential $V_i^{(n)}$ for different neurons i as a function of time. The intial slope is always zero. For $\alpha_i^{(n)} = 1$ slopes of all neurons i are increased by 1 at $t_{\max}^{(n-1)} = t_{\min}^{(n)}$. **c.** For $\alpha_i^{(n)} = 1 - \sum_j W_{ij}^{(n)}$ slopes of all neurons i are equal to 1 for $t \geq t_{\min}^{(n)}$.

Neuron dynamics In the hidden layers and similarly to [Stanojevic et al. (2022b)], the potential $V_i(t)$ of neuron i models an integrate-and-fire neuron. Given the spike times $t_j^{(n-1)}$ of neurons j in the previous layer, the potential $V_i^{(n)}$ of neuron i in layer n follows the dynamics:

$$\tau_c \frac{\mathrm{d}V_i^{(n)}}{\mathrm{d}t} = \alpha_i^{(n)} H(t - t_{\min}^{(n)}) + \sum_j W_{ij}^{(n)} H(t - t_j^{(n-1)}) \tag{3.2}$$

where $t_{\min}^{(n)}$ is a constant which will become by construction a lower bound of the earliest possible spike time in layer n, $\alpha_i^{(n)}$ is a positive scalar which can be seen as the weight of an external spike at $t_{\min}^{(n)}$, $W_{ij}^{(n)}$ is the synapse strength from neuron j to neuron i, and H denotes the Heaviside function which takes a value of 1 for positive inputs and is 0 otherwise. When the potential $V_i^{(n)}$ reaches the threshold $\vartheta_i^{(n)}$, neuron i generates a spike at time $t_i^{(n)}$ and sends it to the next layer. Once a neuron spikes we assume a very large refractory period to ensure that every neuron spikes at most once. According to Eq. (3.2), each input spike $t_j^{(n-1)}$ changes the *slope* of the potential by a fixed amount proportional to $W_{ij}^{(n)}$. Integration of Eq. (3.2) therefore leads to a piecewise linear behaviour of the potential, see Fig. 3.1. Without loss of generality, we assume V to be unit-free and so are the parameters $W_{ij}^{(n)}$ and $\alpha_i^{(n)}$, whereas t and τ_c have units of time. Rescaling time by $t \to (t/\tau_c)$ would remove the units, but we keep it in the equations to show the role of the conversion factor τ_c. Note that this is the same spiking neuron model as defined in [Stanojevic et al. (2022b)] with the minor modification that the ramping input of strength $\alpha_i^{(n)}$ arrives at $t_{\min}^{(n)}$ and not $t_{\min}^{(n-1)}$, which will simplify our equations in the following. We initialize $\vartheta_i^{(n)}$, $t_{\min}^{(n)}$, $t_{\max}^{(n)}$ so that all the neurons of layer n spike once in the interval $[t_{\min}^{(n)}, t_{\max}^{(n)}]$. The threshold $\vartheta_i^{(n)}$ is defined as $\vartheta_i^{(n)} \stackrel{\text{def}}{=} \tilde{\vartheta}_i^{(n)} + D_i^{(n)}$ where $D_i^{(n)}$ is a model parameter initialized at 0 and $\tilde{\vartheta}_i^{(n)}$ is the base threshold. We define a maximum spike time $t_{\max}^{(n)}$, after which emission of a spike is forced in all neurons of layer n which have not spiked yet. The construction of $t_{\min}^{(n)}$ and $t_{\max}^{(n)}$ is recursive as $t_{\min}^{(n)} \stackrel{\text{def}}{=} t_{\max}^{(n-1)}$. Details of the choice of the base threshold and $t_{\max}^{(n)}$ are given in Section 3.6.2.

Adaptive $t_{\max}^{(n)}$ parameters During training as we update the network parameters $W_{ij}^{(n)}$ and $D_i^{(n)}$ the hyperparameters like $t_{\max}^{(n)}$ need to be changed as well such that the condition that all the neurons of layer n spike once in the interval $[t_{\min}^{(n)}, t_{\max}^{(n)}]$ remains true. We suggest a new adaptive update rule which moves $t_{\max}^{(n)}$. Note that this is an addition to the model in [Stanojevic et al. (2022b)] where an adaptive $t_{\max}^{(n)}$ was not necessary since the parameters were fixed. Formally, when processing the training dataset, we update $t_{\max}^{(n)}$ as follows:

$$\Delta t_{\max}^{(n)} = \begin{cases} \gamma(t_{\max}^{(n)} - \min_{i,\mu} t_i^{(n)}) - (t_{\max}^{(n)} - t_{\min}^{(n)}), & \text{if } t_{\max}^{(n)} - t_{\min}^{(n)} < \gamma(t_{\max}^{(n)} - \min_{i,\mu} t_i^{(n)}) \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

The minimum operator iterates over all neurons i and input samples μ in the batch and γ is a constant. After this update, we change the subsequent time window accordingly so that: $t_{\min}^{(n+1)} = t_{\max}^{(n)}$, and we iterate over all layers sequentially. The base threshold $\tilde{\vartheta}_i^{(n)}$ is then updated accordingly, see Section 3.6.2. This adaptation effectively moves all the spikes $t_j^{(n)}$ away from the boundary $t_{\min}^{(n)}$. For simplicity, in the theory section, we consider that this update has reached an equilibrium, so we consider that $t_{\max}^{(n)}$, $t_{\min}^{(n)}$, and $\tilde{\vartheta}_i^{(n)}$ are constants w.r.t. the SNN parameters, the condition $t_{\min}^{(n+1)} = t_{\max}^{(n)}$ is always satisfied and all the spikes of layer n arrive within $[t_{\min}^{(n)}, t_{\max}^{(n)}]$.

Output layer The output layer has index $N+1$ and contains non-spiking read-out neurons. Each neuron m simply integrates input spikes coming from layer N without firing. Integration of $V_m^{(N+1)}$ stops at time $t_{\min}^{(N+1)}$ and the *softmax* and the standard cross-entropy loss $\mathscr{L}$ are calculated using real-valued potentials, analogous to real-valued activations of ANNs.

General reverse mapping from SNN to ANN Building upon the conversion method from ANN with rectified linear units (ReLUs) to TTFS networks [Stanojevic et al. (2022b)], we now describe a reversed mapping strategy defining uniquely the parameters of an equivalent ANN for given SNN parameters. This mapping will be a fundamental pillar in the following theoretical analysis. In the most general case, to find a ReLU network with weights w and bias $\mathbf{b}$ which is equivalent to our SNN model, we define:

$$B_i^{(n)} \overset{\text{def}}{=} \alpha_i^{(n)} + \sum_k W_{ik}^{(n)} \quad \text{and} \quad w_{ij}^{(n)} \overset{\text{def}}{=} \frac{W_{ij}^{(n)}}{B_i^{(n)}} \quad \text{and} \quad b_i^{(n)} \overset{\text{def}}{=} -\frac{\vartheta_i^{(n)}}{B_i^{(n)}} + \frac{t_{\max}^{(n)} - t_{\min}^{(n)}}{\tau_c}. \tag{3.4}$$

Here, $B_i^{(n)}$ has a simple interpretation: it is the slope of the potential at the moment of threshold crossing in the SNN if time is measured in units of τ_c (see Eq. (3.2)). We call $B_i^{(n)}$ the 'slope-at-threshold factor' and it will play an important role in the following. Then if we define the ANN activation at the input layer as the pixel intensity $\mathbf{x}^{(0)}$, Eq. (3.4) defines uniquely a ReLU network with activations $\mathbf{x}^{(n)}$ such that (this is the reciprocal mapping inspired by [Stanojevic et al. (2022b)]):

$$\mathbf{x}^{(n)}\tau_c = t_{\max}^{(n)} - \mathbf{t}^{(n)}. \tag{3.5}$$

At the output layer, we resort directly to the simpler parameter mapping of the output layer from [Stanojevic et al. (2022b)]: with $w_{ij}^{(N+1)} \overset{\text{def}}{=} W_{ij}^{(N+1)}$ and $b_i^{(N+1)} \overset{\text{def}}{=} \alpha_i^{(N+1)}(t_{\max}^{(N)} - t_{\min}^{(n)})$ the logits and the cross-entropy loss $\mathscr{L}$ are also equal in the SNN and the equivalent ANN.

Proof. Starting from the introduced SNN definition, we compute analytically the spikes at time $t_i^{(n)}$ in the SNN. In case the potential $V_i^{(n)}$ reaches the threshold $\vartheta_i^{(n)}$ before $t_{\max}^{(n)}$, the spiking condition $\vartheta_i^{(n)} = V_i^{(n)}(t_i^{(n)})$ yields:

$$\tau_c \vartheta_i^{(n)} = \alpha_i^{(n)}(t_i^{(n)} - t_{\min}^{(n)})H(t_i^{(n)} - t_{\min}^{(n)}) + \sum_j W_{ij}^{(n)}(t_i^{(n)} - t_j^{(n-1)})H(t_i^{(n)} - t_j^{(n-1)}). \tag{3.6}$$

Since we constructed the SNN such that $t_i^{(n)}$ arrives in the time window $[t_{\min}^{(n)}, t_{\max}^{(n)}]$, all the terms $H(\cdot)$ are equal to 1 in the previous equation. So for any spike $t_i^{(n)}$ arriving before $t_{\max}^{(n)}$, we have:

$$t_i^{(n)} = \frac{\tau_c \vartheta_i^{(n)} + \alpha_i^{(n)} t_{\min}^{(n)} + \sum_j W_{ij}^{(n)} t_j^{(n-1)}}{\alpha_i^{(n)} + \sum_k W_{ik}^{(n)}} = \frac{A_i^{(n)}}{B_i^{(n)}} \,. \tag{3.7}$$

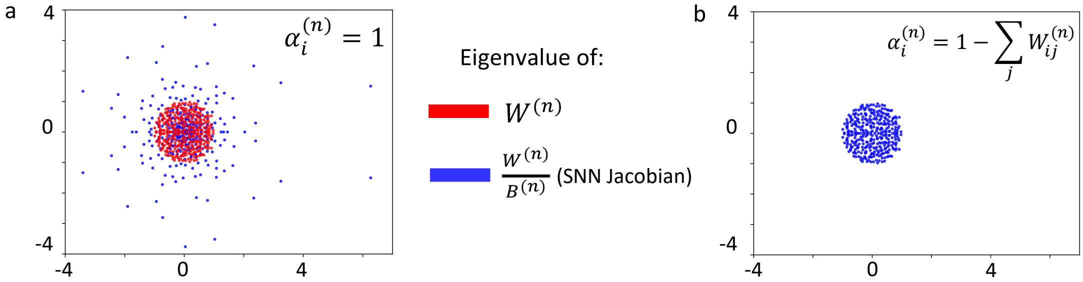


Figure 3.2: **Eigenvalues of the SNN Jacobian under naive initialization and solution. a.** In the general case, when the SNN is initialized with standard deep learning initialization, the eigenvalues of the SNN spread beyond the unit circle. **b.** It is possible to correct for this using the inverse relation between $w^{(n)}$ and $W^{(n)}$. With the linearly mappable condition, this correction is not necessary.

We can already identify the slope-at-threshold B_i in the denominator, then we replace $\alpha_i^{(n)}$ by $B_i^{(n)} - \sum_k W_{ij}^{(n)}$ in $A_i^{(n)}$ and subtract $t_{\max}^{(n)}$ on both sides:

$$t_i^{(n)} - t_{\max}^{(n)} = \tau_c \frac{\vartheta_i^{(n)}}{B_i^{(n)}} + t_{\min}^{(n)} - t_{\max}^{(n)} + \sum_j \frac{W_{ij}^{(n)}}{B_i^{(n)}} (t_j^{(n-1)} - t_{\min}^{(n)}). \tag{3.8}$$

Using this identity, and using that the rectified linear unit is in its operating regime $x_i > 0$ if and only if the spiking neuron i fires before $t_{\max}$, one can now prove by induction that the definition Eq. (3.4) defines an equivalent ReLU network satisfying the identity from Eq. (3.5). □

3.3 Analysis of learning dynamics

The linearly mappable condition We now define a specific choice of SNN, so-called *linearly mappable SNN* and we will show that it satisfies the theoretical conditions for robust SNN training via gradient descent optimization. *The linearly mappable condition* is defined by the choice of $\alpha_i^{(n)}$:

$$\alpha_i^{(n)} = 1 - \sum_j W_{ij}^{(n)}. \tag{3.9}$$

This choice implies that the slope-at-threshold $B_i^{(n)} = 1$, and results in the linear mapping formula:

$$w_{ij}^{(n)} \overset{\text{def}}{=} W_{ij}^{(n)} \quad \text{and} \quad b_i^{(n)} \overset{\text{def}}{=} -\vartheta_i^{(n)} + \frac{t_{\max}^{(n)} - t_{\min}^{(n)}}{\tau_c} \tag{3.10}$$

Vanishing-gradient problem for deep SNNs Previous TTFS-SNNs with exact gradients use mostly shallow networks containing one hidden layer [Göltz et al. (2020); Comşa et al. (2021); Mostafa (2018); Stanojevic et al. (2022a)], or at most 4 hidden layers [Zhang et al. (2021)]. The question arises why the exact gradient approach does not scale to larger networks. We demonstrate in this section that TTFS networks are prone to yield vanishing or exploding gradients (vanishing-gradient problem [Goodfellow et al. (2016); Bengio et al. (1994)]). To solve this problem, we show that a tight balance has to be respected between the weight initialization and the slope-at-threshold vector $\mathbf{B}^{(n)}$. This analysis will result in the definition of robust initialization schemes for TTFS networks.

The vanishing-gradient problem has been studied exhaustively in ANNs [Bengio et al. (1994)]. Similarly, TTFS networks are also subject to this problem. To see this, one has to observe that the network state at layer n is summarized by the vector of spike timings $\mathbf{t}^{(n)}$ such that the loss with respect to the weight parameter at layer n factorizes as:

$$\frac{\mathrm{d}\mathscr{L}}{\mathrm{dW}^{(n)}} = \frac{\mathrm{d}\mathscr{L}}{\mathrm{d}\mathbf{V}^{(N+1)}} \frac{\mathrm{d}\mathbf{V}^{(N+1)}}{\mathrm{d}\mathbf{t}^{(N)}} \frac{\mathrm{d}\mathbf{t}^{(N)}}{\mathrm{d}\mathbf{t}^{(N-1)}} \cdots \frac{\mathrm{d}\mathbf{t}^{(n+1)}}{\mathrm{d}\mathbf{t}^{(n)}} \frac{\mathrm{d}\mathbf{t}^{(n)}}{\mathrm{dW}^{(n)}}. \tag{3.11}$$

where $\mathbf{V}^{(N+1)}$ is a vector containing potentials of neurons in layer $N+1$ at time $t_{\mathrm{min}}^{(N+1)}$. Hence, if the product of Jacobians: $\frac{\mathrm{d}\mathbf{t}^{(n+1)}}{\mathrm{d}\mathbf{t}^{(n)}}$ is naively defined, the amplitude of this gradient might vanish or explode exponentially fast as the number of layers becomes large. As analyzed in [Bengio et al. (1994); Sussillo and Abbott (2014)], a way to solve this problem is to make sure that the largest eigenvalues of the Jacobian $\frac{\mathrm{d}\mathbf{t}^{(n+1)}}{\mathrm{d}\mathbf{t}^{(n)}}$ are close to 1 in absolute value.

We now compute analytically the Jacobian of the SNN. It requires the definition $M_i^{(n)}$ which is 1 if and only if spike $t_i^{(n)}$ arrives before $t_{\mathrm{max}}^{(n)}$. We also denote with $M^{(n)}$ the matrix containing $M_i^{(n)}$ on the diagonal and 0 elsewhere, with $\mathbf{t}^{(n)}$ a vector of spike times in layer n and with $\mathrm{B}^{(n)}$ a diagonal matrix of slope-at-threshold factors. By deriving Eq. (3.7) we find that the Jacobian of the network can be written as ($\cdot$ is the matrix multiplication):

$$\frac{\mathrm{d}\mathbf{t}^{(n)}}{\mathrm{d}\mathbf{t}^{(n-1)}} = M^{(n)} \cdot \frac{1}{B^{(n)}} \cdot W^{(n)} \tag{3.12}$$

We can now analyze the conditions for which the vanishing-or-exploding gradient problems are solved. We observe primarily that (1) the eigenvalues of this Jacobian are strongly determined by the slope-at-threshold $B^{(n)}$ and not only by the weight matrix $W^{(n)}$ as in ANN; then (2) the eigenvalues of the Jacobian of the SNN are the same as the eigenvalues of the Jacobian of the equivalent ANN. To see this, one may recall from Eq. (3.4) that the ANN weights are $w^{(n)} = \frac{1}{B^{(n)}} \cdot W^{(n)}$ and $M_i^{(n)}$ is 1 if and only if the equivalent ReLU unit is saturated.

Robust initialization of TTFS networks Before defining a generic recipe for initializing the weight matrix $W^{(n)}$, we illustrate why using naively the standard deep learning recipes with SNN results in vanishing or exploding gradients. In Fig. 3.2, we demonstrate numerically that this naive approach faces the vanishing-gradient problem. We initialized the weight matrix of an SNN with $W^{(n)} = \frac{1}{\sqrt{340}}\mathscr{N}(0,1)$ where 340 is the number of units in the layers (this is one of the many standard choices in deep learning) so the eigenvalue of $W^{(n)}$ with largest absolute value is close to 1. We can use this matrix to estimate the eigenvalue spectrum of the SNN at initialization. Following classical work in the ANN literature [Sussillo and Abbott (2014); He et al. (2015)], we assume that $M^{(n)}$ has a small impact on the distribution of the

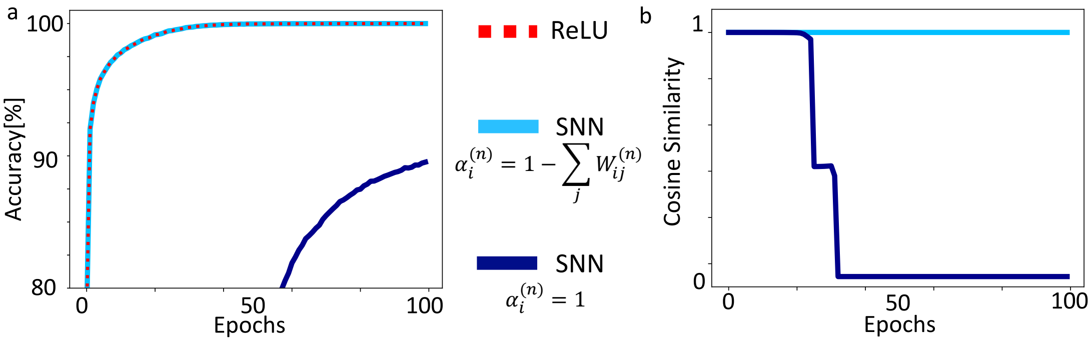


Figure 3.3: **Learning trajectories are biased without the linearly mappable condition. a.** Training of an 8-layer SNN on the MNIST dataset [Deng (2012)] with and without the linearly mappable condition. Both are initialized correctly: we make sure the eigenvalues of the Jacobian at initialization lie within the unit circle. The SNN (light blue) follows the same training curve as the equivalent ANN (red-dashed) only under the linearly mappable condition. **b.** Weights $W^{(n)}$ of the SNN and $w^{(n)}$ of the equivalent ANN remain the same (Cosine Similarity=1) throughout training under the linearly mappable condition, otherwise $\frac{W^{(n)}}{\mathrm{B}^{(n)}}$ is the same as $w^{(n)}$ in the beginning but diverges during training.

eigenvalues, and we can display the eigenvalue spectrum of $w^{(n)} = \frac{1}{B^{(n)}} \cdot W^{(n)}$ which are closely related to the eigenvalue spectrum of the SNN Jacobian $\frac{\mathrm{d}\mathbf{t}^{(n)}}{\mathrm{d}\mathbf{t}^{(n-1)}}$. As shown in Fig. 3.2a, this naive initialization produces multiple eigenvalues with modulus larger than 1 when $\alpha_i^{(n)} = 1$; this eigenvalue spectrum leads to an explosion of the gradient norm in backpropagation.

Our theory also provides a recipe to initialize the SNN outside of the *linearly mappable condition*. Since we know from Eq. (3.12) that a good SNN initialization requires the equivalent ANN to have a good initialization, we can first choose the matrix in the ANN parameter space, and map it to the SNN initialization with the inverse map of Eq. (3.4). This is for instance automatically corrected with the *linearly mappable condition* in Fig. 3.2b since the SNN and ANN are the same, so the eigenvalues stay tightly within the unit circle, showing numerically that the vanishing-gradient problem is avoided.

Biased gradient descent trajectory with the generic mapping Beyond initialization, we also analyze whether the gradient descent trajectory in the SNN parameter space necessarily follows the gradient descent trajectory of the equivalent ReLU network. To describe the gradient descent trajectory of the SNN, we consider a gradient descent step with learning rate η when applying backpropagation to the SNN: $\Delta W_{ij}^{(n)} = -\eta \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{ij}^{(n)}}$, and compute the corresponding update in the space of the ANN parameters. We denote with $\delta w_{ij}^{(n)}$ the update in ANN parameter space, and we use $\mathcal{M}_{\boldsymbol{\alpha}}$ to denote the mapping formula such that $w_{ij}^{(n)} = \mathcal{M}_{\boldsymbol{\alpha}}(W_{ij}^{(n)})$; see Eq. (3.4). If we assume that $\alpha_i^{(n)}$ is a constant independent of $W_{ij}^{(n)}$, we find: $\delta w_{ij}^{(n)} = \mathcal{M}_{\boldsymbol{\alpha}}(W_{ij}^{(n)} - \eta \frac{d\mathcal{L}}{dW_{ij}^{(n)}}) - \mathcal{M}_{\boldsymbol{\alpha}}(W_{ij}^{(n)})$. Assuming a small learning rate, we make a first-order approximation using the derivative $\frac{\mathrm{d}\mathcal{M}_{\boldsymbol{\alpha}}}{\mathrm{d}W_{ij}^{(n)}} = \frac{\mathrm{d}w_{ij}^{(n)}}{\mathrm{d}W_{ij}^{(n)}} = \frac{B_i^{(n)} - W_{ij}^{(n)}}{(B_i^{(n)})^2}$ leading to the approximate update in ANN parameter space:

$$\delta w_{ij}^{(n)} \approx -\eta \frac{\mathrm{d}\mathscr{M}_{\boldsymbol{\alpha}}}{\mathrm{d}W_{ij}^{(n)}} \delta W_{ij}^{(n)} = -\eta \frac{\mathrm{d}\mathscr{M}_{\boldsymbol{\alpha}}}{\mathrm{d}W_{ij}^{(n)}} \frac{\mathrm{d}\mathscr{L}}{\mathrm{d}W_{ij}^{(n)}} = -\eta \left[\frac{\mathrm{d}w_{ij}^{(n)}}{\mathrm{d}W_{ij}^{(n)}} \right]^2 \frac{\mathrm{d}\mathscr{L}}{\mathrm{d}w_{ij}^{(n)}} \tag{3.13}$$

The difference between Eq. (3.13) and a direct ANN update obtained through gradient descent $\delta w_{ij}^{(n)} \propto \frac{\mathrm{d}\mathscr{L}}{\mathrm{d}w_{ij}^{(n)}}$ cannot be corrected with a different learning rate η, because the multiplicative bias in Eq. (3.13) changes for every neuron pair (i, j) and algorithmic iteration. We conclude that in general, the gradient descent trajectory in SNN is "biased", meaning that it is impossible to find a naive gradient descent trajectory in the ANN for which SNN and ANN remain equivalent from initialization to convergence. Under the linearly mappable condition, the multiplicative bias disappears since $\frac{\mathrm{d}w_{ij}^{(n)}}{\mathrm{d}W_{ij}^{(n)}} = 1$ and this is the choice made in Section 3.4. An alternative might be to work with specifically designed 'metrics' [Surace et al. (2020)] that counterbalances the multiplicative factor in Eq. (3.13).

The difficulty to train an SNN with $\alpha_i^{(n)} = 1$ is illustrated in Fig. 3.3. Both SNNs (with or without the linearly mappable condition) are initialized to be equivalent to the same ReLU ANN which solves the vanishing-gradient problem at initialization. Nevertheless, only the SNN with the linearly mappable condition follows the ANN whereas the other one diverges away from the true ReLU trajectories after 20 epochs (Fig. 3.3b). This is true despite using the small learning rate.

3.4 Benchmark results

In the following we always consider SNN initialized with the linearly mappable condition and trained with the Adam optimizer and exponential learning rate schedule (See Section 3.6.3 for simulation details).

Comparison with previous TTFS training We report the performance on the MNIST [Deng (2012)] and Fashion-MNIST [Xiao et al. (2017)] to compare with previous implementation of TTFS training paradigms. We tested a 16-layer fully-connected SNN and a ConvNet SNN (similar to LeNet5). As expected from the theory our network achieves the same performance as the ReLU ANN as seen in Figure 3.4 and Table 3.1. The performance is therefore better than all previous TTFS implementations which were limited to 4 layers (see Section 3.6.3).

Dataset	Acc [%]	
	ReLU	SNN
MNIST	99.57 ± 0.01	99.57 ± 0.00
f-MNIST	94.24 ± 0.02	94.26 ± 0.03
CIFAR10	93.68 ± 0.02	93.69 ± 0.001

Table 3.1: Resulting when training TTFS-SNNs on MNIST, f-MNIST and CIFAR10

State-of-the art TTFS performance on CIFAR100 and PLACES365 Previously, tackling larger-scale image datasets like CIFAR100 [Krizhevsky et al. (2009)] or PLACES365 [Zhou et al. (2017)]

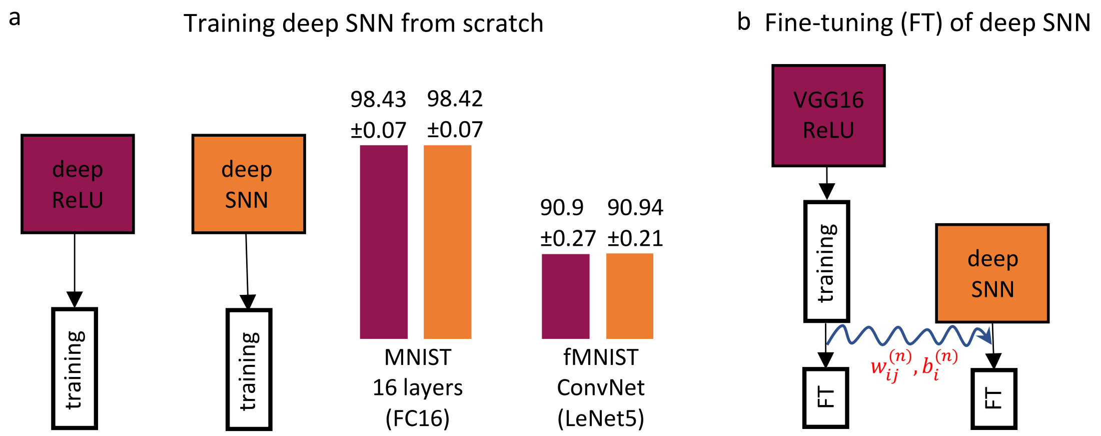


Figure 3.4: **Performance of the SNN under the linearly mappable condition a.** Training deep SNN and ReLU network yields same average performance. **b.** Fine-tuning (FT) of VGG16 SNN network which is initialized with the weights of a pre-trained VGG16 ReLU network. Table 3.2 shows results after fine-tuning for both networks.

(similarly large as ImageNet) was considered impossible. We propose to combine conversion from pre-trained VGG16 and fine-tuning with our approach to build competitive TTFS ConvNet. In Fig. 3.4b we use a pre-trained VGG16 architecture downloaded from an online repository [Geifman (2018); Zhou (2018)] and map it to the SNN without loss of performance (similarly to [Stanojevic et al. (2022b)]). The networks are then fine-tuned with a reduced learning rate. In Table 3.2 we see results for different datasets, fine-tuning provides an increase of the SNN accuracy by 1.76% on CIFAR100 and 1.17% on PLACES365. We are not aware of any TTFS method achieving higher accuracies. More interestingly, fine-tuning SNNs promises to be most useful when the SNN performance is degraded through conversion, for instance, because of hardware constrains like quantization as demonstrated in the next section.

Mitigating quantization, noise effects and reducing latency Let's consider a scenario in which a ReLU network was pre-trained with full precision weights. After mapping to the SNN it is assumed to be deployed on a device with parameter noise, limited temporal resolution or limited weight precision. We use fine-tuning of the VGG16 to recover the SNN accuracy in all these situations in Fig. 3.5. All the experiments are done on CIFAR10 dataset with 10 epochs of fine-tuning. In all three cases (spike time jitter, time-step quantization or SNN

Dataset	Image size	Classes	Acc [%] w/o FT		Acc [%] w/ FT [%]	
			ReLU	SNN	ReLU	SNN
CIFAR100	32 × 32 × 3	100	70.48	70.48	72.23 ± 0.06	72.24 ± 0.06
PLACES365	224 × 224 × 1	365	52.69	52.69	53.86 ± 0.01	53.86 ± 0.02

Table 3.2: VGG16 architecture before and after fine-tuning, for our SNN and the equivalent ANN

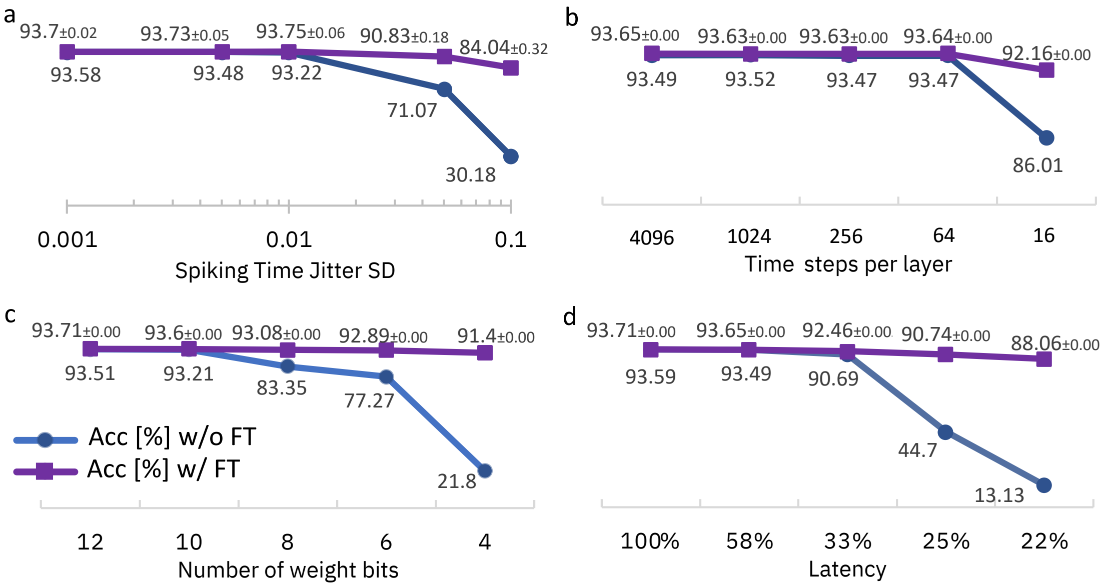


Figure 3.5: **Fine-tuning (FT) VGG16 for quantization/noise robustness or low latency.** In all cases after only 10 epochs of fine-tuning on CIFAR10 (purple) the performance of the initially mapped network (blue) is significantly improved. **a.** Adding noise with certain standard deviation (SD) to all spiking times in the network. **b.** Quantizing spiking times in the network to a given number of time steps per layer. **c.** Representing all weights $W_{ij}^{(n)}$ with given number of bits. **d.** Reducing the latency by reducing the size of $[t_{\mathrm{min}}^{(n)}, t_{\mathrm{max}}^{(n)}]$.

weight quantization) fine-tuning enables to recover the performance. We demonstrate in particular TTFS VGG16 networks achieving higher than 90% accuracy on CIFAR 10 with 16 time-steps per layer or weights quantized on 4 bits. We also investigated whether it is possible to improve the classification latency through fine-tuning by reducing the intervals $[t_{\mathrm{min}}^{(n)}, t_{\mathrm{max}}^{(n)}]$ after conversion from ANN. Doing this we indeed improve latency but the SNN diverges away from the pre-trained ANN. Through fine-tuning, performance higher than 90% test accuracy is recovered, even when the latency is improved by a factor of 4.

3.5 Discussion and future work

In this work we solved the hard instance of a vanishing-gradient problem for single-spike neural networks. Moreover, we showed that through application of the linear-mapping condition the learning trajectories of ReLU ANN and SNN become equivalent. Based on this result, we demonstrated, to the best of our knowledge for the first time, that training of deep single-spike neural networks with sixteen layers yields identical performance as ReLU ANN on large datasets such CIFAR100 and PLACES365. In the future we plan to train even deeper networks and more sophisticated architectures such as ResNets, but this requires to map skip connections to the SNN model which is not trivial.

The work will be probably most impactful when implemented in either digital or analog SNN hardware. We are able to fine-tune the single-spike neural network to adapt the SNN to specific device constraints. Moreover, the learning can be generalized to leaky neuronal dynamics which further addresses the imperfections in hardware elements. After downloading the pre-trained network on the device we envision a continual online learning on chip with energy-efficient and low latency inference. Our demonstration that gradient descent is possible in deep SNNs might provide in the future the opportunities to derive a completely local, hardware-friendly, training algorithm.

3.6 Supplemental information

3.6.1 Generalization to other neuronal dynamics

Linearization of the double exponential In this paper we solve the vanishing-gradient problem for a spiking neural network with piecewise linear postsynaptic potential, i.e. a spike at time $t_j^{(n-1)} < t$ causes a response:

$$a_i^{(n)}(t) = \frac{t - t_j^{(n-1)}}{\tau_c} H(t - t_j^{(n-1)}) \tag{3.14}$$

in neuron i of layer n. However, biologically inspired models in related works [Comşa et al. (2021); Göltz et al. (2021)] often use double-exponential filter, i.e.:

$$a_i^{(n)}(t) = [1 - \exp(-\frac{t - t_j^{(n-1)}}{\tau_1})]\exp(-\frac{t - t_j^{(n-1)}}{\tau_2}) H(t - t_j^{(n-1)}) \tag{3.15}$$

where τ_1 and τ_2 are time constants and $\tau_2 \geq 2\tau_1 > 0$, see. Fig. 3.6.

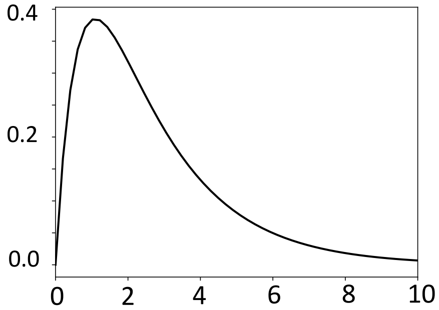


Figure 3.6: An example of $a(t) = [1 - \exp(-\frac{t}{\tau_1})]\exp(-\frac{t}{\tau_2})$ function for $\tau_1 = 1, \tau_2 = 2$.

We notice that in the vicinity of zero, the exponential function can be expressed using Taylor expansion as: $\exp(-\frac{t}{\tau}) = [1 - \frac{t}{\tau} + (1/2)(\frac{t}{\tau})^2 \ldots]$. Therefore the equation for $a_i^{(n)}(t)$ of the double-exponential filter can be approximated around 0 as:

$$a_i^{(n)}(t) = \frac{(t - t_j^{(n-1)})}{\tau_1} H(t - t_j^{(n-1)}), \tag{3.16}$$

Therefore if we replace our SNN model with a double-exponential model we have to define our neuron dynamics with the physically interpretable time constant τ_1 which replaces the time constant τ_c in the neuron dynamics in Eq. (3.2). As a result the voltage $V_i^{(n)}$ of neuron i in layer n at time t becomes:

$$V_i^{(n)}(t) = \alpha_i^{(n)} \frac{(t - t_{\text{min}}^{(n-1)})}{\tau_1} H(t - t_{\text{min}}^{(n-1)}) + \sum W_{ij}^{(n)} \frac{(t - t_j^{(n-1)})}{\tau_1} H(t - t_j^{(n-1)}). \tag{3.17}$$

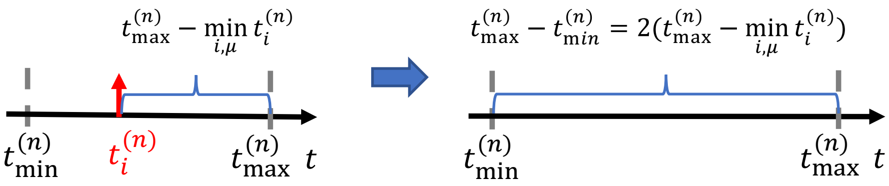

$t_{\max}^{(n)} - \min_{i,\mu} t_i^{(n)}$
$t_{\min}^{(n)}$
$t_i^{(n)}$
$t_{\max}^{(n)}$
t
$t_{\max}^{(n)} - t_{min}^{(n)} = 2(t_{\max}^{(n)} - \min_{i,\mu} t_i^{(n)})$
$t_{\min}^{(n)}$
$t_{\max}^{(n)}$
t

Figure 3.7: **Updating the latest spike time $t_{\max}^{(n)}$ using** $\gamma = 2$. If the minimal spiking time across neurons i and batch inputs μ lies in the first half of the interval $[t_{\min}^{(n)}, t_{\max}^{(n)}]$, the interval is extended.

Scaling the interval $[t_{\min}^{(n)}, t_{\max}^{(n)}]$ to stay in the linear range In order for the network dynamics to always remain in the linear ramping phase of the double-exponential filter, we need to ensure that the linear approximation given in Eq. (3.16) is satisfied within the entire coding interval $\max_n[t_{\max}^{(n)} - t_{\min}^{(n-1)}]$. As the maximum of the function is reached at $t > \tau_1$, see Fig. 3.6, implementing our SNN with a double-exponential model requires the following relationship to be satisfied:

$$\max_n[t_{\max}^{(n)} - t_{\min}^{(n-1)}] < 0.5\tau_1 \tag{3.18}$$

This is possible if we separate the definition of the interval τ_c of the pixel encoding from the neuron time constant τ_1 (see Section 3.6.2 for the recursive construction of the intervals $[t_{\max}^{(n)} - t_{\min}^{(n-1)}]$). Keeping the notation τ_1 for the neuron time constant and τ_c for the pixel encoding, we can construct our network with an arbitrary scaling factor between them to fulfill Eq. (3.18).

Example for a CIFAR 10 network In order to observe what could be concrete values which satisfy the condition given in Eq. (3.18) let's take as an example the CIFAR10 dataset and VGG16 architecture which we have already explored in detail in Section 3.4. For the model which was fine-tuned for reduced latency, see Fig. 3.5d, the $[t_{\min}^{(n-1)}, t_{\max}^{(n)}]$ interval has a value of around $10\tau_c$. Therefore, the acceptable value for τ_1 is $\tau_1 = 20\tau_c$.

Besides being more biologically plausible, the dynamics given with Eq. (3.15) can be impacted by the specifications of the hardware. This can set additional constraints on the time scales of the SNN model and it is likely that fine-tuning the model as explained in the main text would become crucial in this setting.

3.6.2 Setting $t_{\max}^{(n)}$ and the threshold

Initialization: As indicated in the main text, the base threshold $\tilde{\vartheta}_i^{(n)}$ and the parameter $t_{\max}^{(n)}$ are initialized recursively starting from the input interval $[t_{\min}^{(0)}, t_{\max}^{(0)}]$. Let us now assume that we have adjusted the base threshold and the maximum firing time $t_{\max}^{(n-1)}$ up to layer $n-1$.

In layer n, the earliest firing time $t_{\min}^{(n)}$ is defined as: $t_{\min}^{(n)} = t_{\max}^{(n-1)}$. At time $t_{\min}^{(n)}$ we evaluate the membrane potential of all neurons in layer n and determine its maximum $\max_{\mu,i} V_i^{(n)}(t_{\min}^{(n)})$

where the index μ is running over many samples from the training dataset and i iterates over all neurons in layer n. Since for all valid mappings the slope factor $B_i^{(n)}$ of trajectories is positive for $t > t_{\text{min}}^{(n)}$, we define a reference voltage in layer n by

$$\tilde{V}_0^{(n)} = (1+\zeta) \max_{\mu,i} V_i^{(n)}(t_{\text{min}}^{(n)}) \tag{3.19}$$

where $\zeta > 0$ is a small safety margin. We choose the latest possible firing time $t_{\text{max}}^{(n)}$ to be

$$t_{\text{max}}^{(n)} \overset{\text{def}}{=} t_{\text{min}}^{(n)} + \tau_c \tilde{V}_0^{(n)} / B_0 \tag{3.20}$$

where B_0 is a reference slope factor of unit value. We then set the base threshold for neuron i in layer n to

$$\tilde{\vartheta}_i^{(n)} \overset{\text{def}}{=} B_i^{(n)} \left(\frac{t_{\text{max}}^{(n)} - t_{\text{min}}^{(n)}}{\tau_c} \right) \tag{3.21}$$

Since the neuron-specific threshold $\vartheta_i^{(n)} \overset{\text{def}}{=} \tilde{\vartheta}_i^{(n)} + D_i^{(n)}$ is initialized with a shift parameter $D_i^{(n)} = 0$, the choice in Eqs. (3.19) - (3.21) guarantees that, with our initialization of the network parameters, the threshold is reached at a time $t > t_{\text{min}}^{(n)}$ from below.

Note that Eq. (3.21) defines the base threshold for $t > t_{\text{min}}^{(n)}$. Since for $t < t_{\text{min}}^{(n)}$ the membrane potential trajectories could transiently take a value above $\tilde{\vartheta}_i^{(n)}$, we formally set the threshold for $t < t_{\text{min}}^{(n)}$ to a large value (e.g., $100 \cdot \tilde{\vartheta}_i^{(n)}$) so as to make spiking impossible [Stanojevic et al. (2022b)].

For the linear mapping between SNNs and ReLU networks (which is the one chosen to avoid the vanishing-gradient problem) the actual slope factor takes a value $B_i^{(n)} = B_0 = 1$ and this is also the choice for our simulations, i.e., we set $\alpha_i^{(n)} = 1 - \sum_k W_{ik}^{(n)}$. In the context of this mapping, we note that $V_i^{(n)}(t)$ at time $t_{\text{min}}^{(n)}$ has the same value as the activation variable of neuron i in layer n of the equivalent ReLU network, see Eqs. (3.2) and (3.4). Therefore the interval $[t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}]$ is large enough to encode all outputs of layer n in the ReLU network at initialization (with bias parameter initialized at zero).

Iterative updates during training. Throughout training the latest possible firing time and the base threshold are related by Eq. (3.21). In each iteration the parameters $W_{ij}^{(n)}$ and $D_i^{(n)}$ change. Whenever necessary, the iterative update rule for $t_{\text{max}}^{(n)}$ in Eq. (3.3) shifts the maximal firing time in a regime with additional safety margin. This influences in turn the threshold $\vartheta_i^{(n)}$ which is recalculated according to Eq. (3.21).

Additional Remarks.

(i) In principle, we are free to initialize $t_{\text{max}}^{(n)}$ (or $\tilde{\vartheta}_i^{(n)}$) at arbitrarily high values, much larger than those proposed in this subsection. In this case the adaptive rule for $t_{\text{max}}^{(n)}$ (see Fig. 3.7) can be omitted. However, the price to be payed is a very long spiking delay, in particular in networks with many layers.

Both the initialization of the reference voltage with a parameter $\zeta > 0$ and the iterative update rule for $t_{\max}^{(n)}$ in Eq. (3.3) provide a safety margin that lead to spiking delays that could potentially be avoided. However, the $t_{\max}^{(n)}$ used during training does not need to be same as during inference. In order to reduce the classification latency during inference, $t_{\max}^{(n)}$ is recalculated with the fixed parameters found after training such that, in each layer n, the earliest possible spike across all neurons and a representative sample from the training base happens immediately after $t_{\min}^{(n)}$ which in turn leads to a tight value for the threshold via Eq. (3.21).

3.6.3 Simulation details

Each simulation run was executed on one NVIDIA A100 GPU. In all experiments τ_c was set to $1\mathscr{U}$ where $\mathscr{U}$ stands for the concrete unit such as ms or μs. Note that although the choice of units in simulations can be arbitrary, it becomes a critical parameter for a hardware implementation. Moreover, we set $\zeta = 0.5$ whereas a hyperparameter $\gamma = 10$ ensures that even for higher values of initial learning rate all the spikes happen inside the $[t_{\min}^{(n)}, t_{\max}^{(n)}]$ interval. The simulation results were averaged across 16 trials. Batch size was set to 8. In all cases we used the Adam optimizer where for the initial learning rate “lr_0” and iteration “it” an exponential learning schedule was adopted following the formula: $\mathrm{lr}_0 * 0.9^{\frac{\mathrm{it}}{5000}}$.

The data preprocessing included normalizing pixel values to the [0, 1] range and in the case of a fully connected network the input was also reshaped to a single dimension. For MNIST, Fashion-MNIST, CIFAR10 and CIFAR100 the training was performed on the training data whereas the evaluation was performed on the test data. For PLACES365 the fine-tuning was performed on 1% random sample of the training data and the evaluation was performed on the validation data (since the labels for test data are not publicly available).

In fully connected architectures all hidden layers always contain 340 neurons. The LeNet5 contains three convolutional, two max pooling and two fully connected layers with 84 and 10 neurons, respectively. Moreover, some of the datasets utilize a slightly modified version of VGG16. The kernel was always of size 3 and the input of each convolutional operation was zero padded to ensure the same shape at the output. For MNIST and Fashion-MNIST datasets, due to a small image size, the first max pooling layer in VGG16 was omitted. In this case there are two fully connected hidden layers containing 512 neurons each. For CIFAR10 and CIFAR100 the convolutional layers are followed by only one fully connected hidden layer containing 512 neurons, yielding 15 layers in total. Finally, for PLACES365, there are two fully connected hidden layers with 4096 neurons each. The spiking implementation of max pooling operation was done as in [Stanojevic et al. (2022b)].

Analysis of learning dynamics In Fig. 3.3 we illustrated that training SNN with $\alpha_i^{(n)} = 1$ is difficult. For the optimization process we used plain stochastic gradient descent (SGD). In Fig. 3.3a the SNN with linearly mappable condition was trained with initial learning rate equal to 0.0005 which is the same as in the corresponding ReLU network. When $\alpha_i^{(n)} = 1$ the learning process with the same initial learning rate struggles to surpass the training accuracy of around 20%. In this case we found the optimal initial learning rate to be 0.00003, leading to a slower training compared to both ReLU network and SNN with linearly mappable condition. In Fig. 3.3b the goal is to understand how much the SNN weights diverge from the ReLU weights during training. In order to enable a fair comparison in this case all three networks were trained with initial learning rate equal to 0.00003.

Comparison with previous TTFS training approaches A shallow network with one fully-connected hidden layer is tested on MNIST and Fashion-MNIST datasets, similar to networks in [Göltz et al. (2021); Comşa et al. (2021); Zhang et al. (2021); Mostafa (2018); Stanojevic et al. (2022a)]. Before training, the ReLU network and SNN are initialized with the same parameters and the seed is fixed in order to avoid any other source of randomness. Both networks are trained for 50 epochs with initial learning rate equal to 0.0005. In Table 3.3 we see the performance comparison, where all networks contain 340 or more neurons in the hidden layer. Therefore, our model is one of the smallest, but nonetheless its performance surpasses all other approaches.

For deeper networks we noticed that even though the SNN and ReLU network are initialized with the same parameters, they exhibit different performance after certain number of epochs due to numerical instabilities. For this reason, in all cases where the number of hidden layers is larger than one we report the average performance across trials.

Dataset	Model	Acc [%]
MNIST	ReLU	98.3
MNIST[ours]	**SNN**	**98.3**
MNIST [Comşa et al. (2021)]	SNN	97.96
MNIST [Zhang et al. (2021)]	SNN	98
MNIST [Stanojevic et al. (2022a)]	SNN	98.2
fMNIST	ReLU	90.14
fMNIST[ours]	**SNN**	**90.14**
fMNIST [Zhang et al. (2021)]	SNN	88.1
fMNIST [Stanojevic et al. (2022a)]	SNN	88.93

Table 3.3: Performance of shallow fully-connected networks.

State-of-the art TTFS performance on CIFAR100 and PLACES365 Some of the pretrained models we use have batch normalization layers [Geifman (2018)]. The exact mapping fuses them with neighbouring fully connected and convolutional layer similar as in [Stanojevic et al. (2022b)], after which the fine-tuning is conducted for 10 epochs. Since the models are already pretrained, the fine-tuning is done with a reduced initial learning rate of 10^{-6} for CIFAR100 and 10^{-7} for PLACES365. This gives state-of-the-art performance of TTFS-SNN on large datasets. Importantly, the simulations show that the SNN fine-tuning yields zero performance loss compared to the corresponding ReLU network. Therefore when fine-tuning SNN with hardware constraints, we can be sure that the reduced performance comes solely from the device properties and not the training algorithm.

Mitigating quantization, noise effects and reducing latency We consider independently four types of constraints: (i) spiking time jitter, (ii) reduced number of time steps per layer, (iii) reduced number of weight bits, and (iv) limited latency. In practice, these constraints often coexist, and for the future work it would be of interest to consider them not only separately but also jointly.

(i) Spiking time jitter. A Gaussian noise of given standard deviation is added to the spiking times in each layer. This leads to a reduced classification accuracy in comparision with the original noise-free performance, see Fig. 3.5a. Fine-tuning the SNN improves the classification accuracy and especially avoids drastic failure of the network when the larger amount of noise is present.

(ii) Time quantization. In digital hardware, the spike times of the network are subjected to quantization leading to discrete time steps. To mitigate the impact of the spike time outliers, the size of the $[t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}]$ interval is chosen to contain 99% of the activation function outputs in layer n when training data is sent to the input of the ReLU network. This results in the smaller spiking times being "clipped" at $t_{\text{min}}^{(n)}$. We emphasize that in this case the adaptive rule which changes $t_{\text{max}}^{(n)}$ is not applied. I.e. the initial interval is divided into quantized steps which are fixed during the fine-tuning. The network is very robust to discretization until the number of time steps is reduced to 16 where the fine-tuning brings the network within 1.5% of the full-precision performance, see Fig. 3.5b.

(iii) Weights quantization. To reduce the size of the storage memory, we apply quantization aware training such that at the inference time the weights are represented with a smaller number of bits. Similarly as for the spiking time, we remove outliers before the quantization. In this case we remove a predefined percentile as follows on both sides of the distribution. In case of a larger number of bits, only the first and last percentile were removed. However, in case of a 4-bit representation we reduce the interval further by removing first four and last four percentiles. As before, the obtained range is divided into quantized steps which are then fixed during the fine-tuning. At the inference time the quantized steps are scaled to the integer values on $[-2^{q-1}, 2^{q-1}-1]$ range (where q is the number of bits), whereas the other parameters are adjusted accordingly. The fine-tuning leads to a 6-bit representation reaching the performance within 1% of the baseline, whereas in case of 4-bit representation retraining recovers SNN from a complete failure, see Fig. 3.5c.

(iv) Reduced latency. Finally, the robustness to a reduced classification latency is tested by picking smaller $[t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}]$ intervals. We emphasize that the adaptive rule which changes $t_{\text{max}}^{(n)}$ is not applied here. I.e. the initial interval is fixed during fine-tuning. Once again the $[t_{\text{min}}^{(n)}, t_{\text{max}}^{(n)}]$ interval is obtained such that it contains some percentage of the activation function outputs in layer n when training data is sent to the input of the ReLU network. The chosen values of percentiles are 100, 99, 95, 92 and 90 (yielding 22% of the initial latency). This leads to smaller spiking times being "clipped" at $t_{\text{min}}^{(n)}$. For convolutional layers we keep $\zeta = 0.5$,

and otherwise we use $\zeta = 0$. When the latency is reduced to $\approx 33\%$ (95^{th} percentile), the fine-tuning keeps the performance close to 1% of the baseline. Further decrease in latency leads to detrimental reduction in accuracy of the baseline network which is however significantly mitigated by the short SNN fine-tuning, see Fig. 3.5d.

4 Conclusion

In the brain, spikes enable low-cost, robust and long-distance communication. Spiking neural networks intend to replicate some of the same behaviour with the objective of developing a high-performance, energy-efficient, artificial intelligence. This thesis focuses on spiking neural networks with sparse temporal coding schemes. The SNNs are evaluated on data classification tasks using different setups, including supervised learning, conversion from pretrained ANNs, or a combination of both.

In Chapter 2, we explore the question of whether there is an equivalence between a ReLU network and an SNN with TTFS coding and linear postsynaptic potential. It is found that starting from a multi-layer ReLU network, there exists a family of exact mappings that determine the SNN parameters and guarantee an approximation-free conversion. We observe that for a deep ReLU network which can contain fully-connected, convolutional, batch normalizaton and max pooling layers, an equivalent deep SNN achieves the same accuracy on larger benchmarks such as CIFAR100, PLACES365 and PASS. The experiments show that such SNN is robust to spike time jitter, whereas it is more sensitive to scenarios where its parameters are affected by noise. Therefore, a pretrained ReLU network and a mapped SNN are mathematically equivalent, but the performance of the SNN may be affected by device limitations.

Upon analysing the learning in deep single-spike neural networks with linear postsynaptic potential, Chapter 3 identifies an instance of the vanishing-and-exploding gradient problem. It is found that by relying on a specific exact mapping, the SNN parameters can be determined in a way that enables successful training and ensures equivalent learning trajectories between the SNN and ReLU network. We observe that a deep SNN can be trained or fine-tuned to achieve the same performance as the ReLU network on benchmark datasets such as CIFAR10, CIFAR100 and PLACES365. Moreover, the experiments show that the spiking neural network can be efficiently fine-tuned to constraints such as quantization, noise or limited latency. We conclude that deep SNNs with TTFS coding have the potential to be successfully implemented in hardware, providing energy-efficient high-accuracy classification.

The research field as whole continues to investigate the development of low-power, biologically-inspired solutions that can compete with modern AI implementations. With this thesis, we aspire to contribute to the community by considering algorithms which lead to functional *deep spiking models* with *sparse temporal coding*. We see our work as evidence that deep and sparse spiking neural networks can have the same performance as rate-coded SNNs or standard ANNs. Furthermore, the presented solutions demonstrate a certain level of robustness when confronted with device limitations. In this regard, we hope that our research will inspire potential hardware implementations, allowing for comprehensive testing of various metrics ranging from performance and energy consumption to silicon area.

A Additional publications

A.1 Approximating ReLU networks by single-spike computation

Paper information

Authors: Ana Stanojevic, Evangelos Eleftheriou, Giovanni Cherubini, Stanisław Woźniak, Angeliki Pantazi, Wulfram Gerstner

Abstract Developing energy-saving neural network models is a topic of rapidly increasing interest in the artificial intelligence community. Spiking neural networks (SNNs) are biologically inspired models that strive to leverage the energy efficiency stemming from a long process of evolution under limited resources. In this paper we propose a SNN model where each neuron integrates piecewise linear postsynaptic potentials caused by input spikes and a positive bias, and spikes *maximally* once. Transformation of such a network into the ANN domain yields an approximation of a standard ReLU network, leading to a facilitated training based on backpropagation and an adaptation of the batch normalization. With backpropagation-trained weights, SNN inference offers a sparse-signal and low-latency classification, which can be readily adapted for a stream of input patterns, lending itself to an efficient hardware implementation. The supervised classification of MNIST and Fashion-MNIST datasets, using this approach, provides accuracy close to that of an ANN and surpassing other single-spike SNNs.

Keywords spiking neural network, one spike per neuron, image processing, ReLU, efficient classification

Publication The chapter is accepted as a conference paper at IEEE International Conference

on Image Processing (ICIP) 2022. ([Stanojevic et al. (2022a)])

A.1.1 Introduction

Even though the processing of information in the brain is highly sophisticated, it does not require a large amount of electrical power for its operation. Artificial neural networks (ANNs) adopt a neuron model, that was initially inspired by biological neurons [McCulloch and Pitts (1943)]. However, the advances in modern-day artificial intelligence (AI) are mainly driven by abstract neuronal units [Goodfellow et al. (2016)]. This progress was often targeting the accuracy of prediction on benchmark tasks, with limited consideration of energy efficiency [Bianco et al. (2018)]. Therefore, it is of great importance to *go back* to the principles of the brain's operation to better understand and implement concepts leading to sustainable AI, where also energy efficiency and classification latency are important metrics.

Spiking neural networks (SNNs) are closer to biology than ANNs. They assume spike-based neuronal communication and stateful neuronal dynamics. However, it is still unclear which algorithm is the best for training SNNs. The brain possesses complex learning mechanisms that have been studied for decades [Kandel et al. (2000); Markram et al. (2011); Frémaux and Gerstner (2016)] and applied to SNNs [Zenke et al. (2015)], but are often unable to achieve the performance of ANNs trained with backpropagation. Due to non-differentiable nonlinearities in the SNN neuronal dynamics, a backpropagation algorithm that relies on derivatives to solve the optimization problem related to learning is not directly applicable. There has been extensive research to adapt the backpropagation through time (BPTT) algorithm to the SNN training [Huh and Sejnowski (2018); Gardner et al. (2015); Woźniak et al. (2020); Neftci et al. (2019)] and to derive more biologically plausible approximations [Bellec et al. (2020); Zenke and Ganguli (2018); Neftci et al. (2019)]. Some of these approaches proved to be successful in solving benchmark problems with classification accuracy close to that of ANNs. However, these models often use a large number of spikes by encoding information in the rate of neuronal activity. Rate coding indeed complies with early neuroscience experiments [Hubel and Wiesel (1959)], in which it has been observed that certain stimuli cause particular neurons in the brain to increase their spiking rate. However, it also introduces a significant classification latency.

Further experiments [Kubke et al. (2002); Gollisch and Meister (2008); Johansson and Birznieks (2004); O'Keefe and Recce (1993)] suggested that in some parts of the brain information is encoded in the precise timing of individual spikes. Building on this insight, SNN training mechanisms were recently introduced, where each neuron spikes exactly once [Göltz et al. (2021); Comşa et al. (2021); Mostafa (2018); Kheradpisheh and Masquelier (2020); Zhang et al. (2021)]. In this paper we focus on pattern-classification applications and propose an SNN model, where each neuron spikes at most once, thus improving system efficiency. Specifically, each neuron has an associated observation period, after which it is assumed that the membrane potential is so low that no further spiking is possible. The inputs are represented according to the time-to-first-spike principle, where important information is encoded into earlier spike times and the predicted class is determined from the output neuron that spikes first. The neuronal dynamics are described by a non-leaky integrate-and-fire

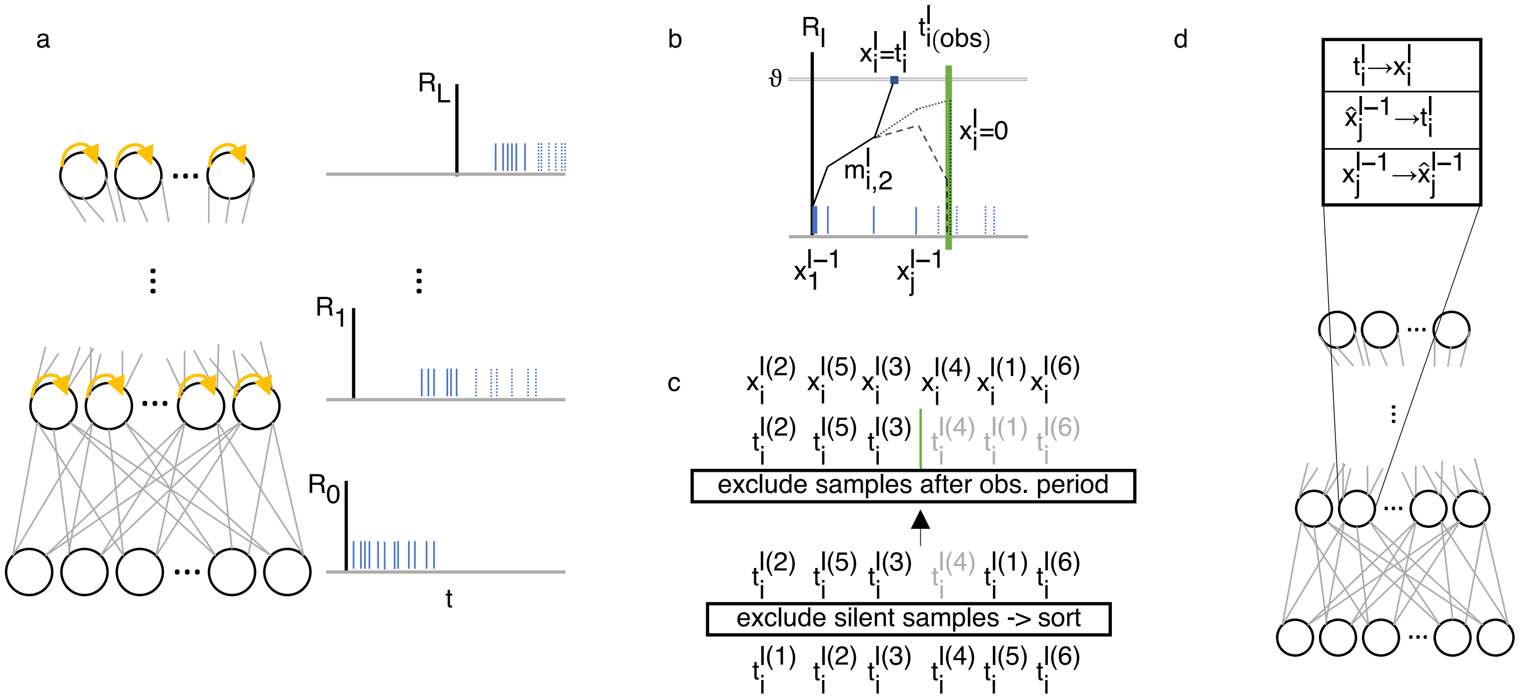

a
R_L
R_1
R_0
t
b
R_l
ϑ
x_i^l=t_i^l
t^l_i(obs)
x_i^l=0
m^l_i,2
x_1^l−1
x_j^l−1
c
x_i^l(2) x_i^l(5) x_i^l(3) x_i^l(4) x_i^l(1) x_i^l(6)
t_i^l(2) t_i^l(5) t_i^l(3) t_i^l(4) t_i^l(1) t_i^l(6)
exclude samples after obs. period
t_i^l(2) t_i^l(5) t_i^l(3) t_i^l(4) t_i^l(1) t_i^l(6)
exclude silent samples -> sort
t_i^l(1) t_i^l(2) t_i^l(3) t_i^l(4) t_i^l(5) t_i^l(6)
d
t_i^l→x_i^l
x̂_j^l−1→t_i^l
x_j^l−1→x̂_j^l−1

Figure A.1: Mapping of SNN to an equivalent ANN a) SNN network with stateful IF neurons. Each layer $0 \leq l \leq L$ has a reference time R_l b) Dynamics of membrane potential V_i^l in three different situations where each neuron has an observation time leading to a subset of "late" spikes (after the green vertical line) to be removed c) Processing flow of spike times $t_i^{l(b)}, 1 < b < B$ of one batch of size $B = 6$ d) Equivalence of the SNN with an ANN having stateless neurons

model with a positive integration bias and a linear postsynaptic potential. The SNN can be transformed into an equivalent feed-forward ANN network, where neurons communicate through their spiking times. The derivatives are directly calculated with respect to the spiking times, as required by the temporal coding scheme. Furthermore, the linear postsynaptic potential leads to a piecewise linear response in the feed-forward network, approximating a ReLU model. The observation period is chosen to reflect the bias and batch normalization is used as in ReLU networks, yielding faster training and increased accuracy. During inference, the SNN resorts to very few spikes exhibiting low latency, as a neuron typically integrates only a subset of the inputs that are received prior to its spiking time.

A.1.2 Method

We consider a spiking neural network with L layers, where each layer $1 \leq l \leq L$ contains N^l non-leaky integrate-and-fire neurons and there are N^0 input neurons. The membrane potential of neuron i in layer $l \geq 1$ evolves according to $\frac{dV_i^l}{dt} = \alpha_i^l + I_i^l$, where I_i^l is the total synaptic current arriving at neuron i in layer l and $\alpha_i^l > 0$ denotes a positive bias. The real-valued information Γ_j presented at neuron j of the input layer, is encoded into a spike with spiking time:

$$x_j^0 = t_j^0 = \frac{\Gamma_{\max} - \Gamma_j}{\Gamma_{\max}} T + \epsilon \tag{A.1}$$

where $\Gamma_{\max}$ is the maximum real-valued input in the database of input patterns and T is the time period over which the input spikes are sent to the SNN. Specifically, the earliest spike occurs at $x_j^0 = \epsilon = 10^{-4}$ with the reference time $R_0 = 0$, whereas $x_j^l = 0$ denotes the lack of spiking events. The reference time R_l of layer l is determined as the arrival of the first spike from the previous layer, i.e., $R_l = \min_{k:x_k^{l-1} \neq 0}(x_k^{l-1})$ (see Fig. A.1a). R_l is considered the beginning of neuronal events from the perspective of layer l and defines the time instant when the bias term $\alpha_i^l t$ is activated (see Eq. A.2). The incoming spiking times in layer l are perceived as $\hat{x}_j^{l-1} = (x_j^{l-1} - R_l + \epsilon) H(x_j^{l-1})$, where H denotes the Heaviside function. Each spike triggers a linearly increasing postsynaptic potential in the receiving neuron. The membrane potential $V_i^l(t)$ of neuron i in layer l is expressed as:

$$V_i^l(t) = \alpha_i^l t + \omega H(t - t_{i(\text{obs})}^l) + \sum_{j=1}^{N^{l-1}} w_{ij}^l H(t - \hat{x}_j^{l-1}) H(\hat{x}_j^{l-1})(t - \hat{x}_j^{l-1}) \tag{A.2}$$

where $t_{i(\text{obs})}^l$ indicates the observation period after which no incoming spike is processed by the neuron, ω represents a negative constant with large absolute value and w_{ij}^l defines the synaptic strength between neurons j and i. If the membrane potential reaches the threshold ϑ at time t_i^l, a spike is generated, i.e. $x_i^l = t_i^l$ (see Fig. A.1b). Otherwise, $x_i^l = 0$ denotes the lack of spiking and it will be ignored in the calculations of the next layer. We assume that the neuron has a long refractory period and therefore it can generate at most one spike per input pattern.

Mapping from SNN to ANN

The processing of the proposed SNN network can be mapped to that of a specific feed-forward ANN network, where the spiking information is communicated as the real values x_i^l. We now map spike times t_i^l in the SNN to the variable x_i^l in the ANN (see Fig. A.1d). If the membrane potential $V_i^l(t)$ reaches the threshold ϑ then we have:

$$\vartheta = \alpha_i^l t_i^l + \sum_{j \in J_i^l} w_{ij}^l H(\hat{x}_j^{l-1})(t_i^l - \hat{x}_j^{l-1}) \tag{A.3}$$

where J_i^l denotes the set of input spikes that were received before spiking time t_i^l, i.e., $H(t_i^l - \hat{x}_j^{l-1}) > 0$. Therefore, the spiking time of neuron i in layer l is:

$$t_i^l(J_i^l) = \frac{\vartheta + \sum_{j \in J_i^l} w_{ij}^l \hat{x}_j^{l-1}}{m_i(J_i^l)} \tag{A.4}$$

where $m_i^l(J_i^l) = \sum_{j \in J_i^l} w_{ij}^l H(\hat{x}_j^{l-1}) + \alpha_i^l$ is the slope of the membrane potential after receiving the last input spike in set J_i^l (see Fig. A.1b). The membrane potential doesn't reach the threshold ϑ if either the slope $m_i^l(J_i^l)$ is negative (silent neuron) or the observation time $t_{i(\mathrm{obs})}$ has already elapsed. In order to account for this if-condition, we introduce an auxiliary variable $s_i^l(J_i^l)$:

$$s_i^l(J_i^l) = H(m_i^l(J_i^l))H(t_{i(\mathrm{obs})} - t_i^l) \tag{A.5}$$

where the first Heaviside function denotes that the membrane potential reaches the threshold with positive slope, whereas the second one enforces that this occurs before the expiration of the observation period. Furthermore, to account for the assumed refractoriness in SNNs, the set J_i^l is defined as the *smallest* subset of input spikes, such that $s_i^l(J_i^l) > 0$ and where the next input spike (if there is one) comes after time $t_i^l(J_i^l)$. If such a subset doesn't exist, we assume that J_i^l contains all input spikes. Finally, the output of neuron i in layer l is defined as:

$$x_i^l = t_i^l(J_i^l) s_i^l(J_i^l) \tag{A.6}$$

where the neuron outputs $x_i^l = t_i^l$ if there exist J_i^l such that condition $s_i^l(J_i^l) = 1$ is satisfied, and 0 otherwise. Eqs. A.4 and A.6 together implement a ReLU nonlinearity.

Observation period approximates batch normalization

The neuronal membrane potential exhibits a non-linear behavior due to the finite observation period, the silent neuron condition (see Eqs. A.5 and A.6) or the causality condition $H(t - \hat{x}_j^{l-1})$ (see Eq. A.4). We focus on the first cause. To determine $t_{i(\mathrm{obs})}^l$ we adopt ideas from ANN ReLU networks. In ReLU networks the bias value determines the threshold between zero and non-zero outputs, whereas batch normalization ensures that the inputs effectively utilize the operation range on both sides. Here we adjust the batch threshold statistics by finding the median and tuning the observation time so that each neuron spikes 50% of the time. During training, for the observed neuron i and batch size B, we would like to ensure that the neuron outputs a non-zero value for $\frac{B}{2}$ samples (see Fig. A.1c). Let us denote the number of samples that lead to a silent neuron i in layer l as $\zeta_i^l \leq \frac{B}{2}$. The spiking times are then sorted, and $t^l_{i(\mathrm{obs})}$ is determined such that the $\frac{B}{2} - \zeta_i^l$ largest spiking times are mapped to zero. Furthermore, the moving average of $t_{i(\mathrm{obs})}$ is stored for each neuron during training and is used during the inference phase.

A.1.3 Results

We assess the SNN performance on permutation invariant versions of two image recognition benchmark datasets: MNIST and Fashion-MNIST. MNIST contains images of handwritten digits, while Fashion-MNIST comprises images of fashion items, both labeled with 10 possible classes. Images have dimensions 28 × 28 and are split into 10000 test images and 60000 used for training and validation in 9:1 ratio.

The weights w_{ij}^l are initialized with a uniform distribution over the interval $[-\frac{1}{\sqrt{8N^{l-1}}}, \frac{1}{\sqrt{8N^{l-1}}}]$ and $\alpha_i^l = 1$. The training is performed using backpropagation with a log-loss function and a softmax activation function with hyperparameter β:

$$x_m^L = \frac{e^{-\beta t_m^L}}{\sum_{m=1}^{N^L} e^{-\beta t_m^L}} \qquad \mathscr{L} = -\sum_{m=1}^{N^L} y_m log(x_m^L) \tag{A.7}$$

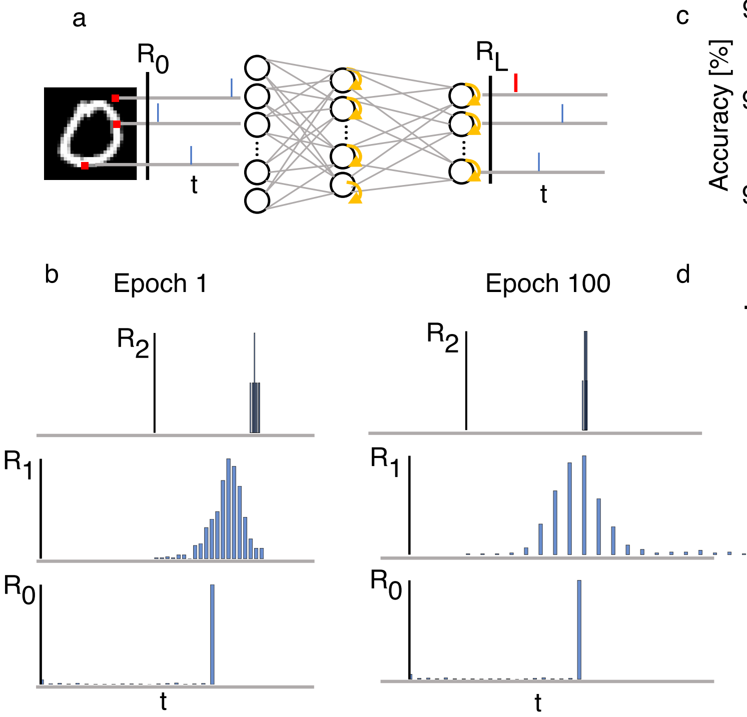
where y_m denotes the one-hot encoded labels. To avoid a zero value argument in the log function during training, the updates are ignored when there is at least one silent neuron in the last layer. Furthermore, two penalty terms are added to the loss function of Eq. A.7:

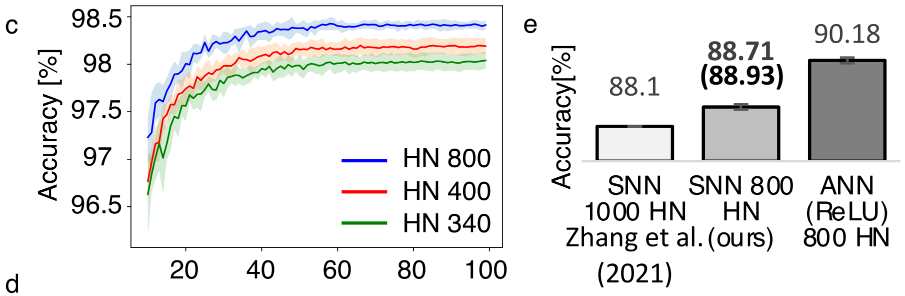
$$\mathscr{L}_{\mathrm{silent}} = \kappa \sum_i \max(0, -\sum_j w_{ij}^l) \tag{A.8}$$

$$\mathscr{L}_{\mathrm{reg}} = \lambda \sum_{i,j} (w_{ij}^l)^2 \tag{A.9}$$

where κ and λ are hyperparameters. The small absolute values of the weights that are obtained through initialization and kept during training with the regularization loss (Eq. A.9) make α_i^l the dominant factor in determining the spiking time distribution and diminish the non-linearity stemming from causality $H(t - \hat{x}_j^{l-1})$. The regularizer in Eq. A.8 ensures that the summation of input weights to a neuron doesn't drop below zero so as to avoid silent neurons.

Training was performed using the Adam [Kingma and Ba (2014)] optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$), an exponential learning rate schedule (learning rate=0.0002, 500 decay steps and 0.96 decay rate) and a batch size of 64. Furthermore, we applied gradient clipping with value a 1000, while dropout was used to increase generalization. Other hyperparameters of the system are: $T = 1$, $\vartheta = 1$, $\beta = 1000$. The class is predicted by the index of the earliest spiking neuron (see Fig. A.2a), and classification accuracy is reported.





d

	Model	Arch.	Accuracy[%] (max)
Ours	ANN (ReLU)	800 HN	98.44±0.08
Ours	**SNN**	**800 HN**	**98.41±0.05 (98.5)**
Ours	**SNN**	**400 HN**	**98.19±0.09 (98.39)**
Ours	**SNN**	**340 HN**	**98.03±0.09 (98.2)**
Zhang et al. (2021)	SNN	800 HN	98.5
Zhang et al. (2021)	SNN	400 HN	98.1
Comșa et al. (2021)	SNN	340 HN	97.96
Illing et al. (2019)	SNN	5000 HN	98.6±0.1
Mostafa (2018)	SNN	800 HN	97.55

Figure A.2: Results for MNIST and Fashion-MNIST datasets a) MNIST images are encoded and fed into a shallow SNN. The predicted class is determined as the index of the neur. on that fires first (red) b) Distributions of spiking times; the big vertical bar in the input layer corresponds to black pixels c) Test accuracy for different SNN architectures for MNIST dataset d) MNIST test accuracy for different number of hidden neurons (HN) e) Fashion-MNIST test accuracy, the max. accuracy is in parenthesis

To compare with earlier work [Comşa et al. (2021); Mostafa (2018); Zhang et al. (2021)], the MNIST dataset is tested with three shallow architectures of 800, 400 or 340 hidden neurons. The results in Figs. A.2c and A.2d show that our SNN system well approximates the ANN network performance with ReLU non-linearity. Furthermore, the performance of our system surpasses state-of-the-art single-spike models across different architectures. In Fig. A.2e we observe that for the more challenging Fashion-MNIST dataset SNN comes close to the ANN performance and outperforms the previous state-of-the-art result for single-spike models.

Discussion

The proposed neuronal dynamics of our model retains the biological spirit, but it is simplified as incorporating more biological effects would significantly increase the complexity of approximating a ReLU network. Unlike in ReLU networks, where the bias can be trained, in our model a neuron is trained to transmit spikes to the next layer in about 50% of the cases. Furthermore, in a ReLU network the batch normalization uses the mean value and the variance of the batch distribution to shift and scale the neuronal output. In our case we use a median and there is no scaling of spike distribution.

A one-spike-per-neuron network typically uses significantly fewer spikes than a rate-coded network, therefore it increases energy efficiency. Furthermore, for classification tasks the class is determined after the first output neuron spikes, yielding a short inference latency. For a network with 800 hidden neurons, classification of images from the test set occurs after receiving on average 233 spikes from the hidden layer (see Fig. A.2b). This paper advances the state of the art by reducing the overall number of spikes as each neuron spikes *at most* once while increasing the classification accuracy by leveraging techniques inspired by ReLU networks. Our approach is also applicable to convolutional layers, making it a research milestone towards mapping complex image processing networks onto efficient SNNs.

A.1.4 Conclusion

We propose an SNN model, where each neuron spikes at most once and the input spikes together with a positive integration bias determine a piecewise linear postsynaptic potential. An observation period limits the number of spikes and leads to an SNN that is equivalent to an ANN with ReLU activations. Such equivalence enables training within AI frameworks originally developed for the ANN domain, while inference in the SNN domain is performed with a low number of spikes and low latency. Importantly, in hardware realizations temporally encoded visual input spike patterns could be supplied directly from the output of a retina-like chip [Gallego et al. (2020); Lichtsteiner et al. (2008)]. The classification accuracy achieved on two image datasets is close to the ANN baselines and surpasses previously reported single-spike SNN performance.

A.2 Time-encoded multiplication-free spiking neural networks: application to data classification tasks

Paper information

Authors: Ana Stanojevic, Giovanni Cherubini, Stanisław Woźniak, Evangelos Eleftheriou

Abstract Spiking neural networks (SNNs) are mimicking computationally powerful biologically inspired models in which neurons communicate through sequences of spikes, regarded here as sparse binary sequences of zeros and ones. In neuroscience it is conjectured that time encoding, where the information is carried by the temporal position of spikes, is playing a crucial role at least in some parts of the brain where estimation of the spiking rate with a large latency cannot take place. Motivated by the efficiency of temporal coding, compared with the widely used rate coding, the goal of this paper is to develop and train an energy-efficient time-coded deep spiking neural network system. To ensure that the similarity among input stimuli is translated into a correlation of the spike sequences, we introduce correlative temporal encoding (CTE) and extended correlative temporal encoding (ECTE) techniques to map analog input information into *input spike patterns*. Importantly, we propose an implementation where all multiplications in the system are replaced with at most a few additions. As a more efficient alternative to both rate-coded SNNs and artificial neural networks (ANNs), such system represents a preferable solution for the implementation of neuromorphic hardware. We consider data classification tasks where *input spike patterns* are presented to a feed-forward architecture with leaky-integrate-and-fire (LIF) neurons. The SNN is trained by backpropagation through time with the objective to match sequences of output spikes with those of specifically designed *target spike patterns*, each corresponding to exactly one class. During inference the *target spike pattern* with the smallest van Rossum distance from the *output spike pattern* determines the class. Extensive simulations indicate that the proposed system achieves a classification accuracy at par with that of state-of-the-art machine learning models.

Keywords spiking neural network, temporal coding, low-complexity data classification, multiplication-free inference, backpropagation through time

Publication The chapter is accepted as a journal paper at Neural Computing and Applications 35 (9) ([Stanojevic et al. (2023a)]).

Acknowledgments We thank Dr. Angeliki Pantazi for her insightful comments and suggestions.

Conflict of interest The authors declare that they have no conflict of interest.

Data availability The datasets utilized during the current study are available in public repositories.

A.2.1 Introduction

The human brain is a phenomenal system capable of mapping sensory inputs to many non-trivial responses in a highly computationally efficient manner. Many studies of biological neural structures have been conducted to investigate how the information from sensory inputs is processed to obtain a desired response. For example, the structure used to process scents in the fruit fly olfactory system has been studied to determine the computational strategies that take place in the insect's brain for solving similarity search problems [Dasgupta et al. (2017); Eichler et al. (2017); Ryali et al. (2020)]. One of the most debated questions in artificial intelligence is whether neural networks are able to mimic their biological counterparts when solving problems of practical interest. Nowadays, classification tasks are found in ubiquitous applications, from edge devices as part of the emerging internet-of-things network [Jankowski et al. (2020)], to servers where data storage at large scale requires classification of items in terms of relevance or category [Cherubini et al. (2016)]. Data classification is rather well understood in the machine learning and artificial neural network (ANN) communities [Cortes and Vapnik (1995); Schmidhuber (2015)]. However, ANNs usually require floating-point multiplications between the synaptic weights and the activations, making them quite inefficient in terms of power consumption [Strubell et al. (2020); Devlin et al. (2018)]. Inspired by the computationally efficient brain, spiking neural networks (SNNs) constitute a class of neural networks [Ghosh-Dastidar and Adeli (2009); Ponulak and Kasinski (2011); Zambrano et al. (2019); Boybat et al. (2018); Sebastian et al. (2018)], where neuronal information is communicated asynchronously through sequences of spikes. The spike trains are commonly interpreted as sequences of sparse binary signals, leading to an efficient operation and are therefore of importance for many applications [Garain et al. (2021); Luo et al. (2022); Toğaçar et al. (2021)]. Due to their low-power operation, SNNs are often considered together with their hardware implementation. Some of the most important challenges in this domain include designing specific hardware components [Liu et al. (2021)] and online learning rules [Hu et al. (2021)] as well as autonomous mobile robots control based on SNNs [Cao et al. (2015); Jimenez-Romero and Johnson (2017)].

The problem of energy efficiency has long been recognized in the machine learning community. In order to increase the energy efficiency of ANNs, binarized neural networks (BNNs) [Courbariaux et al. (2016)] have been proposed where only bit-wise operations are executed by constraining all the activations and weights to +1 and -1 values. While extremely efficient, this approach results in a performance gap which is to some extent addressed by different techniques trying to reduce the quantization error [Lin et al. (2020); Liu et al. (2020); Wang et al. (2021)] or improve training [Lin et al. (2021); Bulat and Tzimiropoulos (2019); Liu et al. (2018); Xu et al. (2021)]. As a common characteristic, both SNNs and BNNs adopt binary communication, and also largely reduce the required computation and storage. However, spiking neural networks further imply specific neuronal and synaptic dynamics, offering a computationally powerful system which we leverage in our solution.

Newly proposed SNN systems and learning algorithms are often compared with ANNs that exhibit well-established performance [Bellec et al. (2020); Zenke and Ganguli (2018); Comşa et al. (2021)]. However, an overall theoretical framework for the supervised training of SNNs has not been developed yet, mainly because the computation of gradients for weight updates requires differentiation of functions with discontinuities. Early approaches to SNN training include converting a pre-trained ANN into an SNN representing continuous ANN communication as spiking rates [Rueckauer et al. (2017)], introducing differentiable approximations of non-continuous functions [Huh and Sejnowski (2018)], and resorting to a probabilistic framework [Gardner et al. (2015)]. Training schemes that avoid taking the derivative of functions with discontinuities were also presented [Comşa et al. (2021); Hunsberger and Eliasmith (2016)]. Recently, it was proposed to train SNNs by a backpropagation through time (BPTT) algorithm, where non-continuous functions are approximated by differentiable functions only during the backward pass [Woźniak et al. (2020); Neftci et al. (2019); Bohte (2011)].

A further active research topic is the encoding of information for neuronal transmission in SNNs. In early neuroscientific experiments [Hubel and Wiesel (1959)], it was observed that a stimulus is able to generate a large number of spikes in some neurons, leading to the conclusion that a high spiking rate is related to a high neuronal response. Rate coding is a well established encoding scheme and many works rely on it for a successful SNN training [Bellec et al. (2020); Woźniak et al. (2020); Neftci et al. (2019)]. In [Fabre-Thorpe et al. (1998)], it was shown that image classification in the brain takes place so fast, i.e., in about 150ms after presenting the stimulus, that there is no sufficient time to reliably estimate the spiking rate for a single neuron. Later neuroscience experiments further confirmed that the temporal position of spikes contains the information [Kubke et al. (2002); Gollisch and Meister (2008); Johansson and Birznieks (2004)]. Therefore, recent work on SNNs is also focused on temporal coding schemes which enable faster inference and require fewer spikes. Specifically, the time-to-first-spike (TTFS) method is an extreme version of temporal coding where each neuron uses only one spike per neuron. TTFS is able to achieve relatively high classification accuracy for simple tasks and shallow architectures, however it requires a new training framework to be developed for each neuronal dynamics and results are often obtained only for the continuous domain [Bohte et al. (2002); Mostafa (2018); Comşa et al. (2021)].

In this paper, we introduce a novel temporal encoding approach for the encoding of input information into *input spike patterns*, whereby the similarity among input stimuli is translated into a correlation of the spike sequences. Two encoding methods based on this approach are investigated: correlative temporal encoding (CTE) and extended correlative temporal encoding (ECTE), which differ in the assumed input data type. Furthermore, we propose a data classification system based on an SNN structure that maps input information into the temporal position of spikes, and determines the network response from the observation of the spike sequences at the output neurons. We also adopt temporal encoding to map class labels into *target spike patterns.* For class prediction we utilize the van Rossum metric, which takes into account the entire sequence of spikes when assessing which *target spike pattern* is closest to the *output spike pattern.* The SNN dynamics [Gerstner et al. (2014)] are trained in a supervised manner, following the recently introduced approach based on spiking neural units

(SNUs) [Woźniak et al. (2020)]. The training is performed by BPTT assuming a weighted binary cross-entropy loss function, with the objective that the sequences of output spikes match those of the *target spike patterns*. Finally, we introduce a multiplication-free implementation of the model, which is obtained by an approximation of the neuronal dynamics by using shift-and-add operations [Marimuthu et al. (2010)], and investigate the possible trade-off between classification accuracy and computational complexity of the proposed system.

As it has been shown in [Woźniak et al. (2020)], it is possible to train deep spiking neural networks with rate coding and reach accuracy close to that of ANNs on many benchmarks. The main purpose of this work is to leverage the brain-inspired dynamics and powerful training setup of a rate coding scheme for high accuracy, while proposing temporal coding schemes to reduce the latency and the number of used spikes. Importantly, reducing the number of spikes together with the multiplication-free implementation of a SNN lead to a superior energy-efficient solution compared to networks based on rate coding as well as standard ANNs, making it a viable alternative for neuromorphic hardware implementations.

A.2.2 System Architecture

Data classification using ANNs usually includes the flow of {preprocessing → network operations → class prediction}. In SNNs, there are further requirements of providing information to the input neurons as sequences of spikes, and of obtaining the predicted class from the sequences of spikes at the output neurons. Therefore, when non-spiking datasets are used, two additional steps need to be considered in the pipeline, i.e., the encoding of input information into spatio-temporal *input spike patterns* and the encoding of class labels into spatio-temporal *target spike patterns*, see Fig. A.3a. In this section, we assume that the two encoding functions are known, and focus on the remaining parts of the system. The input and output encoding functions will be described in detail in Section A.2.3.

Preprocessing

We assume a classification system operating with input data items comprising features extracted from any dataset, e.g., image, text, video, etc. In this paper, we consider datasets with image and text files.

Definition 1. *The set of all input data items is denoted by* $\mathscr{D}$*;* $\mathscr{D}_{train}$ *denotes subset of* $\mathscr{D}$ *containing items for training the classification system.*

Definition 2. *The set of all class labels for the items in the dataset* $\mathscr{D}$ *is denoted by* C*, whereas the set of all features obtained from preprocessing of the dataset* $\mathscr{D}$ *is denoted by* F*. Furthermore, the* i*-th class in the set* C *is denoted by* c_i*.*

The input features $f \in F$ can take any form. For instance, categorical features may be encoded with any pattern rather than with the commonly used one-hot-encoding transformation. To

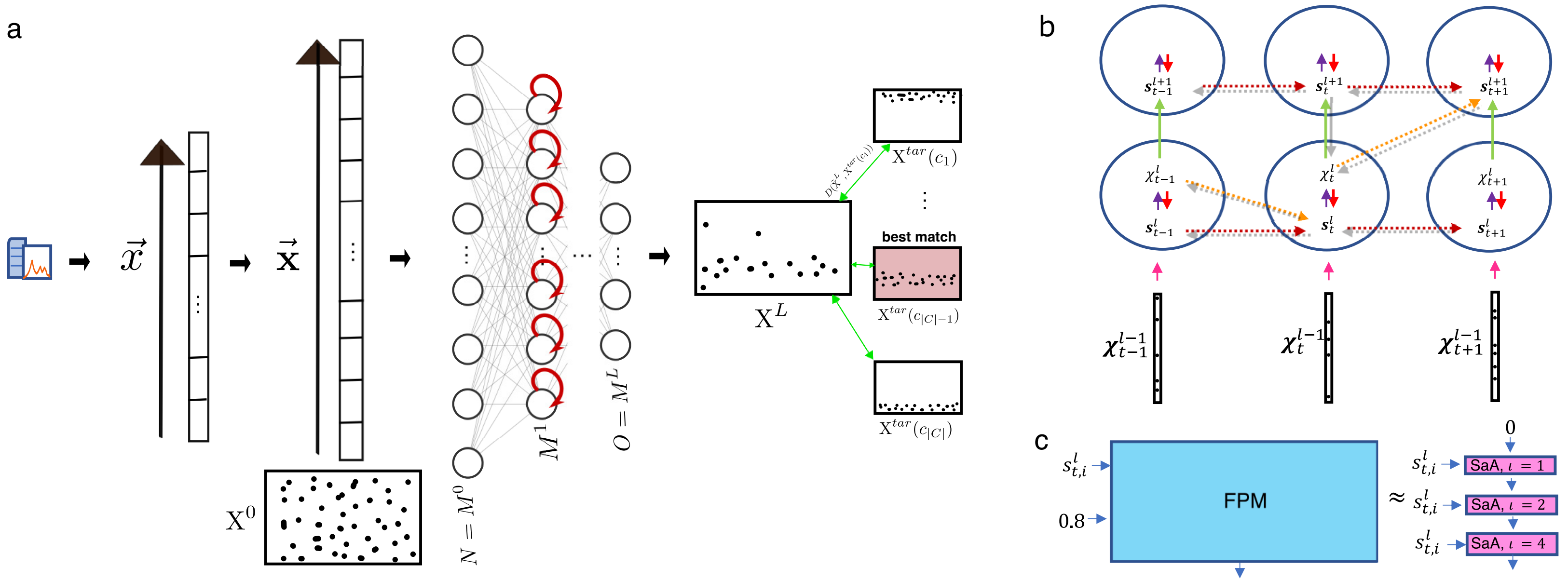


Figure A.3: Architecture of the proposed classification system a) Input items from $\mathscr{D}$ (see Def. 1) go through initial preprocessing to generate real-valued feature vectors; the values of each feature are allocated into q_f bins to generate $\vec{x}$; $\vec{x}$ is then encoded into the binary vector $\vec{\mathbf{x}}$ that is reshaped into the spatio-temporal *input spike pattern* X^0 (dots denote spikes in the diagram); the SNN receives X^0 and generates the *output spike pattern* X^L; the output is compared with the *target spike patterns* corresponding to different classes using the van Rossum metric; the closest pattern finally determines the predicted class b) Computational graph of forward and backward pass c) An example of replacing a full precision multiplication with shift-and-add (SaA) operations, where at each step $s^l_{t,i}$ is shifted by ι positions to the right and added to the previous value

obtain a tractable number of values for numerical features, the values of each numerical feature in training data are uniformly quantized into Q levels, where Q is a system parameter. For each numerical feature f, the quantized values are allocated in q_f bins, where $q_f \leq Q, \forall f$.

Definition 3. *For a feature $f \in F$, $A_f = \{a_{f,1}, a_{f,2}, \ldots, a_{f,q_f}\}$ is the set of quantized values that can be assigned to that feature. Furthermore, we assume that $a_{f,1} \leq a_{f,2}, \ldots, \leq a_{f,q_f}$, where $a_{f,i}$ denotes the i-th value of the feature f.*

A feature f may also take a zero value. In order to increase the efficiency of the classification system, a zero value of a feature will be encoded with a zero signal at the input of the SNN.

Definition 4. *For an input item $x \in \mathscr{D}$, $F_{\neq 0}(x) \subseteq F$ denotes the set of non-zero features.*

We assume that the sets F and $A_f, \forall f$, are finite, and that each input item takes values for all $|F|$ features, with $|F_{\neq 0}(x)| > 0, \forall x$, where $|\cdot|$ denotes the cardinality of a set. An item x can thus be represented as

$$\vec{x} = [x_1, x_2, \ldots, x_{|F|}] \tag{A.10}$$

where $x_i \in A_i$, for $i = 1, \ldots, |F|$ (see Figs. A.3a, A.4a). We remark that the vector notation with an arrow above is used for sequences of features. In other cases, vectors are denoted by the upright boldface type notation.

Spiking Neural Network

We consider SNNs with a feed-forward architecture, i.e., without recurrent connections, with N input neuronal units, $L-1$ fully-connected hidden layers each with $M^l, l = 1 \ldots L-1$, units, and O output neuronal units. The number of units in the input and output layer are sometimes also denoted as M^0 and M^L respectively (see Fig. A.3a). For each input item the SNN receives an *input spike pattern*, which is a binary matrix of dimension $N \times T$, where T is the number of discrete time steps over which the input pattern is defined (see Fig. A.3a). For all the neurons in hidden layers, a leaky-integrate-and-fire (LIF) model is employed [Gerstner et al. (2014)]. At every time instant t, the neurons integrate the binary inputs (0 or 1) that are modulated with the synaptic weights, and store the result into an internal state variable called membrane potential. The rate of decay of the membrane potential is determined by a *leak* parameter $0 < \nu < 1$, see Eq. A.11. When the membrane potential $s^l_{t,i}$ of the i-th neuron in a layer l with M^l units reaches a threshold value β, the neuron generates a spike equal to one, and in all other time instants it outputs zero. The behavior of the M^l neurons receiving inputs from M^{l-1} neurons is described in discrete time by [Woźniak et al. (2020)]:

$$\mathbf{s}^l_t = g(\mathrm{W}^l \cdot \boldsymbol{\chi}^{l-1}_t + \nu \mathbf{1}_{M^l} \odot \mathbf{s}^l_{t-1} \odot (1 - \boldsymbol{\chi}^l_{t-1})) \tag{A.11}$$

$$\boldsymbol{\chi}^l_t = \theta(\mathbf{s}_t - \beta \mathbf{1}_{M^l}) \tag{A.12}$$

for $l = 1, ..., L-1$, where $\mathbf{s}_t^l$ is the M^l-dimensional neuronal state vector at time t, $\boldsymbol{\chi}_t^{l-1}$ and $\boldsymbol{\chi}_t^l$ denote the input and output vectors of layer l at time t, respectively, W^l represents an $M^l \times M^{l-1}$ synaptic weight matrix, $\mathbf{1}_M^l$ is an M^l-dimensional vector with M^l elements all equal to one, and $\odot$ stands for the element-wise product. The activation function θ in our work is set to the *Heaviside function* for all layers, whereas g is equal to the *identity*. The proof that the neuronal dynamics described in discrete time by Eqs. A.11 and A.12 follow the dynamics of a LIF neuron in continuous time is given in [Woźniak et al. (2020)].

The output layer consists of simple integrating neurons with a sigmoid (σ) as an activation function, i.e. $\boldsymbol{\chi}_t^L = \sigma(\mathrm{W}^L \cdot \boldsymbol{\chi}_t^{l-1} - \beta \mathbf{1}_M)$. Each of the $|C|$ possible labels corresponds to a *target spike pattern*, given by a binary matrix of dimensions $O \times T$. During the training phase, at each time instant and for each output neuron a weighted binary cross-entropy loss (see Eq. A.13) is computed between the desired prediction value of the correct class c_i defined by the *target spike pattern* $\mathrm{X}^{\mathrm{tar}}(c_i)$ and the *output spike pattern* X^L (with $\boldsymbol{\chi}_t^l$ as columns, see Fig. A.3a).

$$Loss(\mathrm{X}^L, \mathrm{X}^{\mathrm{tar}}(c_i)) = -\sum_o^O \sum_{t=1}^T \omega X_{o,t}^{\mathrm{tar}}(c_i) log(X_{o,t}^L) + ((1 - X_{o,t}^{\mathrm{tar}}(c_i)) log(1 - X_{o,t}^L)) \tag{A.13}$$

where $X_{o,t}^L$ is the output of neuron with index o at time t, $X_{o,t}^{\mathrm{tar}}(c_i)$ is a corresponding target of class c_i and the constant factor $\omega \geq 1$ is introduced to enhance the contribution of sparse spikes. All the synaptic weights and the threshold β are updated by backpropagation through time (see Fig. A.3b), whereas the leak parameter ν is not trainable. To avoid the singularity arising from the non-differentiability of the *Heaviside* function, an approximation obtained by the derivative of the *tanh* function is adopted, as introduced in [Woźniak et al. (2020)].

We remark that here we focus on a supervised learning approach. A more directly biologically inspired version of the proposed classification system, where unsupervised learning is implemented by a spike-time-dependent-plasticity (STDP) learning rule [Bi and Poo (2001)] and a winner-takes-all approach, was presented in [Stanojevic et al. (2020)].

Class Prediction

During inference, the network is expected to output spikes at time positions that are close to the spike positions in the pattern of the correct target class.

We adopt the van Rossum distance [van Rossum (2001); Gardner et al. (2015); Zenke and Ganguli (2018)] as a distance metric between two spike patterns:

$$\tilde{\mathrm{X}}_{o,t}^L = \tilde{\mathrm{X}}_{o,t-1}^L e^{-\frac{1}{\tau_m}} + \mathrm{X}_{o,t}^L \tag{A.14}$$

$$D(\tilde{\mathrm{X}}^L, \mathrm{X}^{\mathrm{tar}}(c_i)) = \frac{1}{\tau_m} \sum_o^O \sum_{t=1}^T (\tilde{\mathrm{X}}_{o,t}^L - \tilde{X}_{o,t}^{\mathrm{tar}}(c_i))^2 \tag{A.15}$$

where τ_m is a time constant parameter and $\tilde{X}_{o,t}^L$ is equivalent to the convolution of the output spike sequence with an exponential having time constant τ_m. Therefore, the spiking activity of the output neurons is observed over a number of time steps to determine the similarity of spike sequences. The overall distance between output and target patterns is obtained by averaging over all the output neurons. For each input item, the predicted class is determined as the *target pattern* with the smallest van Rossum distance from the *output spike pattern* (see Fig. A.3a).

Multiplication-Free Inference System

A significant advantage of SNNs is the adoption of binary sequences of spikes for neuron communications, i.e., discrete-time signals having 1 if a spike is present or 0 otherwise. This means that the matrix-vector product in Eq. A.11 that would usually require floating point multiplications (FPM) is replaced by a small number of additions (ADD), i.e.,

$$\mathrm{W}^l \cdot \boldsymbol{\chi}_t^{l-1} = \sum_{j:\chi^{l-1}{}_{t,j} \neq 0} W_{ij}^l \tag{A.16}$$

for neuron i in layer l receiving spikes at time t. However, floating point multiplications are present in other parts of the system that might dominate the complexity. Therefore, we propose an implementation where all multiplications are replaced with one or few additions.

There are two other parts of the system where multiplications are performed, namely in the computation of the neuron dynamics in hidden layers (see Eq. A.11) and at the output where sigmoid function and van Rossum distance metric (see Eqs. A.14 and A.15) are calculated. In each time step a membrane potential decay is computed in Eq. A.11 by multiplying the state variable $\mathbf{s}_t$ with a decay parameter ν. To compute the van Rossum metric in Eq. A.14, the convolution value at each step requires multiplying the value at the previous step with the factor $e^{-\frac{1}{\tau_m}}$. Furthermore, the square of the difference of two sequences is required in Eq. A.15. To obtain a computationally efficient realization, we replace the multiplications by the factors ν and $e^{-\frac{1}{\tau_m}}$ with shift-and-add operations (see Fig. A.3c). Eqs. A.11 and A.14 are then expressed as

$$s_{t,i}^l = g\Big(\sum_{j:\chi^l{}_{t,j} \neq 0} \mathrm{W}_{ij}^l + \sum_{\iota} s_{t-1,i}^{l(\iota)} (1 - \chi_{t-1,i}^l)\Big), i = 1, \dots, M^l \tag{A.17}$$

$$\tilde{\mathrm{X}}_{o,t}^l = \sum_{\iota} \tilde{\mathrm{X}}_{o,t-1}^{l(\iota)} + \mathrm{X}_{o,t} \tag{A.18}$$

where $s_{t-1,i}^{l(\iota)}$ and $\tilde{\mathrm{X}}_{o,t-1}^{l(\iota)}$ are obtained from $s_{t-1,i}^l$ and $\tilde{\mathrm{X}}_{o,t-1}^l$, respectively, by multiplication with $2^{-\iota}$, which is equivalent to a shift of ι positions to the right in two's complement binary representation. The square norm in Eq. A.15 is then replaced with the sum of absolute values, yielding

$$D(\tilde{\mathrm{X}}^L, \mathrm{X}^{\mathrm{tar}}(c_i)) = \frac{1}{\tau_{vr}} \sum_{o}^{O} \sum_{t=1}^{T} |\tilde{\mathrm{X}}_{o,t}^L - \tilde{X}_{o,t}^{\mathrm{tar}}(c_i)| \tag{A.19}$$

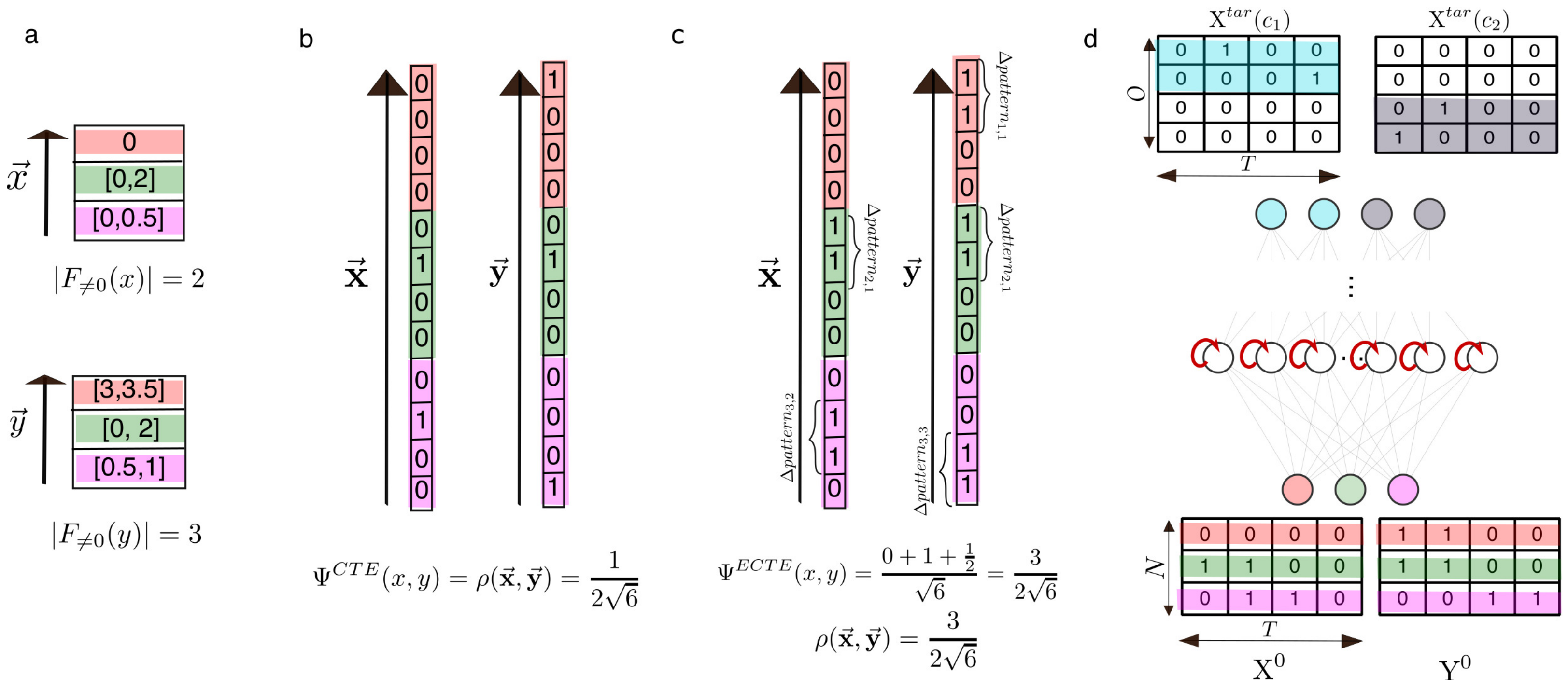


Figure A.4: An example of generating *input spike patterns* for $|F| = 3$, $T = 4$, $N_f = 1$, $d_f = 1$ and *target spike patterns* for $|C| = 2$, $O_c = 2$, $d_c = 1$ a) Two items x and y are represented with sequences of features $\vec{x}$ and $\vec{y}$ which consist of binned values and zeros b) CTE algorithm transforms sequences of features $\vec{x}$ and $\vec{y}$ into $\mathbf{\vec{x}}$ and $\mathbf{\vec{y}}$; Zero value is transformed into zero-vector c) ECTE algorithm transforms sequences of features $\vec{x}$ and $\vec{y}$ into $\mathbf{\vec{x}}$ and $\mathbf{\vec{y}}$ with hyperparameters $Q = 3$, $\eta = 1$, $\delta_n = 1$, $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 2$ d) $\mathbf{\vec{x}}$ and $\mathbf{\vec{y}}$ vectors are reshaped into *input spike patterns* of shape $N \times T$ and sent to SNN; Output of SNN is compared with two *target spike patterns*, where neurons corresponding to the class are active and others are silent

Finally, the computation of the sigmoid function is accomplished by look-up tables, where the quantized values of the sigmoid are stored with a predefined precision. As a further observation, multiplications in the computation of the neuronal dynamics can be completely avoided by adopting the integrate-and-fire (IF) neuronal model [Gerstner et al. (2014)] instead of LIF. However, here we consider the LIF model to allow higher flexibility of the neuronal dynamics.

A.2.3 Time Encoding of Input and Target Patterns

Correlative Time Encoding (CTE)

An objective of input encoding in SNNs is to generate *input spike patterns* while retaining the information about item similarity that is present in the sequences of features. In fact, if for each item an individual random *input spike pattern* were generated, the network would not be able to find any similarity between items in the training dataset and in the test set. It would then be very difficult to perform classification tasks. It is therefore necessary to devise a method for the generation of the *input spike patterns* that retains the similarity between different items and generalizes well whenever new data is encountered. Furthermore, from a biological perspective, it can be conjectured that the signal in the brain carries the information about the similarity of the newly received input with the already processed inputs, thus easing the learning process [Dasgupta et al. (2017)]. In previous work on temporal encoding of input information in SNNs, the spike positions were chosen randomly [Gardner et al. (2015); Zenke and Ganguli (2018)]. In this section, we introduce temporal encoding methods, which yield spike patterns that retain input item similarity.

Let us consider a similarity metric $\psi_f(x_f, y_f)$ between two features x_f and y_f in the sequences of features $\vec{x}$ and $\vec{y}$. We define the similarity between two items x and y as a weighted sum of the similarities for each feature [Venkatesan et al. (2018)]:

$$\Psi_{\boldsymbol{\alpha}}(x, y) = \sum_{f=1}^{|F|} \alpha_f \psi_f(x_f, y_f) \tag{A.20}$$

where $\boldsymbol{\alpha} = [\alpha_1, ..., \alpha_{|F|}]$ indicates the vector of positive real parameters that represent the relevance of the corresponding features and normalize the summation. Here we assume $\alpha = \alpha_f = \frac{1}{\sqrt{|F_{\neq 0}(x)||F_{\neq 0}(y)|}}, \forall f$.

For the correlative time encoding (CTE) algorithm the partial similarity is defined as:

$$\psi_f^{CTE}(x_f, y_f) = \begin{cases} 1, & x_f = y_f, x_f \neq 0, y_f \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{A.21}$$

Therefore, (A.20) becomes:

$$\Psi^{CTE}(x,y) = \alpha \sum_{f, x_f = y_f, x_f \neq 0, y_f \neq 0} 1. \tag{A.22}$$

Thus the similarity between items x and y is determined by the number of equal non-zero features, see Fig. A.4a.

As an *input spike pattern* contains only zeros and ones, it can be associated with a binary vector $\vec{\mathbf{x}}$ of length $n = NT$, obtained by concatenating the binary sequences given by the spike patterns of each neuron. The temporal encoding of input information can thus be seen as a map that relates a sequence of features $\vec{x}$ to a binary vector $\vec{\mathbf{x}}$ (see Fig. A.4b). Two *input spike patterns* that have spikes on the same positions will be perceived by the network as similar, as the same information will propagate for both inputs. Therefore, we choose the cosine similarity as a metric for the similarity between *input spike patterns* representing items x and y:

$$\rho(\vec{\mathbf{x}},\vec{\mathbf{y}}) = \frac{<\vec{\mathbf{x}},\vec{\mathbf{y}}>}{\|\vec{\mathbf{x}}\|\,\|\vec{\mathbf{y}}\|} \tag{A.23}$$

where $< \cdot, \cdot >$ denotes the inner product and $\|\cdot\|$ is the two-norm. The *input spike patterns* will thus exhibit a similarity proportional to the number of spikes in the same positions. The encoding function should map the feature sequence $\vec{x}$ into the binary vector $\vec{\mathbf{x}}$ such that:

$$\forall x, y, \quad \Psi^{CTE}(\vec{x},\vec{y}) \approx \rho(\vec{\mathbf{x}},\vec{\mathbf{y}}). \tag{A.24}$$

Definition 5. *Let* $B = \{\mathbf{b}^{(j)} \in \{0,1\}^{\xi}, 1 \leq j \leq \xi\}$, $\xi \in \mathbb{N}$. *A vector* $\mathbf{b}^{(j)}$, $\forall j$, *is generated as a Poisson process with vanishingly small rate* $\lambda = o(\xi^{-\frac{1}{2}})$; *therefore the average number of ones in* $\mathbf{b}^{(j)}$ *is expressed as* $d = \lambda\xi = o(\xi^{\frac{1}{2}})$, *leading to the generation of sparse vectors. For large but finite* ξ, *we refer to the vectors in* B *as quasi-orthogonal vectors.*

Lemma 1. *Let* $B = \{\mathbf{b}^{(j)} \in \{0,1\}^{\xi}, 1 \leq j \leq \xi\}$ *be a set of quasi-orthogonal vectors. Then* $\forall i, j$, *with* $i \neq j$, $P(<\mathbf{b}^{(i)},\mathbf{b}^{(j)}> \neq 0) \to 0$ *and the vectors* $\mathbf{b}^{(1)},\ldots,\mathbf{b}^{(\xi)}$ *converge in probability to an orthogonal set as* $\xi \to \infty$.

Proof. The inner product of vectors $\mathbf{b}^{(i)}$ and $\mathbf{b}^{(j)}$ can be rewritten as a sum:

$$<\mathbf{b}^{(i)},\mathbf{b}^{(j)}> = \sum_{k=1}^{\xi} b_k^{(i)} b_k^{(j)}.$$

Therefore,

$$P(<\mathbf{b}^{(i)},\mathbf{b}^{(j)}> \neq 0) = P(\sum_{k=1}^{\xi} b_k^{(i)} b_k^{(j)} \neq 0).$$

Let E_k denote the event $b_k^{(i)} b_k^{(j)} \neq 0$. Then

$$P(\sum_{k=1}^{\xi} b_k^{(i)} b_k^{(j)} \neq 0) = P(\cup_{k=1}^{\xi} E_k).$$

From the union bound we have $P(\cup_{k=1}^{\xi} E_k) \leq \sum_{k=1}^{\xi} P(E_k)$. Therefore,

$$P(\sum_{k=1}^{\xi} b_k^{(i)} b_k^{(j)} \neq 0) \leq \sum_{k=1}^{\xi} P(b_k^{(i)} b_k^{(j)} \neq 0).$$

Furthermore, as the vectors are generated independently, we have $P(b_k^{(i)} b_k^{(j)} \neq 0) = P(b_k^{(i)} \neq 0)P(b_k^{(j)} \neq 0)$. Then

$$P(\sum_{k=1}^{\xi} b_k^{(i)} b_k^{(j)} \neq 0) \leq \sum_{k=1}^{\xi} P(b_k^{(i)} \neq 0)P(b_k^{(j)} \neq 0).$$

For large ξ, a Poisson distribution converges to a binomial distribution with parameters ξ and $p = \frac{d}{\xi}$ [Simons and Johnson (1971)], where $d = o(\xi^{\frac{1}{2}})$ and $\forall i, P(b_k^{(i)} \neq 0) = \frac{d}{\xi}$. From the definition of little-o notation we have that $d = o(\xi^{\frac{1}{2}})$ means that $\lim_{\xi \to \infty} \frac{d}{\xi^{\frac{1}{2}}} = 0$. Therefore:

$$\lim_{\xi \to \infty} P(\sum_{k=1}^{\xi} b_k^{(i)} b_k^{(j)} \neq 0) \leq \lim_{\xi \to \infty} \sum_{k=1}^{\xi} P(b_k^{(i)} \neq 0)P(b_k^{(j)} \neq 0) \tag{A.25}$$

$$\sim \lim_{\xi \to \infty} \xi \frac{d}{\xi}\frac{d}{\xi} \tag{A.26}$$

$$= \lim_{\xi \to \infty} \frac{d}{\xi^{\frac{1}{2}}} \lim_{\xi \to \infty} \frac{d}{\xi^{\frac{1}{2}}} = 0 \tag{A.27}$$

□

We now describe the *input spiking pattern* encoding function. For the N input neurons and $|F|$ features, where $N \geq |F|$, we assume that the number of input neurons of the SNN is uniformly divisible by the number of features, although in general a non-uniform assignment is possible. Therefore, each feature is assigned $N_f = N/|F|$ input neurons. For each value $a_{f,i}$ we define a mapping into a binary vector of length $n_f = N_f \times T$.

Definition 6. *For a given non-zero value $a_{f,i}$, a sparse binary vector $\mathbf{a}_{f,i}$ of length $n_f = N_f \times T$ is generated as a Poisson process with rate $\lambda_f = o(n_f^{-\frac{1}{2}})$ ($d_f = \lambda_f n_f = o(n_f^{\frac{1}{2}})$), and λ_f is independent of $a_{f,i}$. The zero value is encoded with a zero vector of length n_f. The set of all vectors $\mathbf{a}_{f,i}$ that may represent a feature f is denoted as $\mathbf{A}_f$.*

From Eq. (A.10), we have that an item x is represented by $\vec{x} = [x_1, x_2, \dots x_{|F|}]$, where $x_1 \in A_1$, $x_2 \in A_2$, etc. The binary vector with length $n = NT$ of the *input spike pattern* associated with item x is given by:

$$\vec{\mathbf{x}} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{|F|}] \tag{A.28}$$

where $\mathbf{x}_1 \in \mathbf{A}_1, \ldots, \mathbf{x}_{|F|} \in \mathbf{A}_{|F|}$ (see Fig. A.4b). The *input spike pattern* X^0 represented as a matrix of dimensions $N \times T$ is then obtained by reshaping the vector $\vec{\mathbf{x}}$ as shown in Fig. A.4d.

As the set F is finite, $n_f \to \infty$ as $n \to \infty$. Therefore, we have that for all non-zero values $a_{f,i}$, $\|\mathbf{a}_{f,i}\|$ converges to $\sqrt{d_f}$ as $n \to \infty$, as all the components are generated as Poisson processes with parameter d_f. Consequently, $\forall x$, $\|\vec{\mathbf{x}}\| = \sqrt{|F_{\neq 0}(x)| d_f}$ as $n \to \infty$.

Lemma 2. *Consider vectors* $\vec{\mathbf{x}}$ *and* $\vec{\mathbf{y}}$ *that are built by concatenating vectors* $\mathbf{x}_f \in \mathbf{A_f}$ *and* $\mathbf{y}_f \in \mathbf{A_f}$, $f = 1, \ldots, |F|$, *respectively. Then* $\rho(\vec{\mathbf{x}}, \vec{\mathbf{y}}) = \frac{<\vec{\mathbf{x}}, \vec{\mathbf{y}}>}{\|\vec{\mathbf{x}}\| \|\vec{\mathbf{y}}\|} = \frac{\sum_f <\mathbf{x}_f, \mathbf{y}_f>}{\sqrt{|F_{\neq 0}(x)||F_{\neq 0}(y)| d_f}}$ *as* $n \to \infty$.

Following the steps of the proof for Lemma 1, one finds that the binary vectors of length n_f that are generated as Poisson processes with parameter d_f, where $d_f = o(n_f^{\frac{1}{2}})$, are quasi-orthogonal. Therefore, the non-zero vectors $\mathbf{a}_{f,i}$ are also quasi-orthogonal and their inner product vanishes as $n \to \infty$. Hence, the inner product of binary vectors corresponding to different features also vanishes. The inner product between two binary vectors $\mathbf{x}_f$ and $\mathbf{y}_f$ for feature f is given in the limit by:

$$< \mathbf{x}_f, \mathbf{y}_f >= \begin{cases} d_f, & x_f = y_f, x_f \neq 0, y_f \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{A.29}$$

Therefore, for the binary vectors $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$, the cosine similarity is given in the limit by:

$$\begin{aligned} \forall x, y, \quad \rho(\vec{\mathbf{x}}, \vec{\mathbf{y}}) &= \alpha \sum_{f, x_f = y_f, x_f \neq 0, y_f \neq 0} \frac{d_f}{d_f} \\ &= \Psi^{CTE}(\vec{x}, \vec{y}). \end{aligned} \tag{A.30}$$

The similarity of the vectors associated with the *input spike patterns* converges as $n \to \infty$ to the similarity among input items (see Fig. A.4b). Simulation results presented in Section A.2.4 indicate that the proposed approach yields satisfactory results for large but finite dimensional vectors.

Extended CTE (ECTE)

So far we have considered the general case, where features can be of any data type. Assuming, as in most applications, that all features are of numerical type, we consider a further similarity metric expressed as:

$$
\begin{aligned}
&\psi_f^{ECTE}(x_f, y_f) = \\
&\psi_f^{ECTE}(a_{f,i}(x_f), a_{f,j}(y_f)) = \\
&\quad\begin{cases} \max(0, \frac{\mu_f - |j-i|}{\mu_f}), x_f \neq 0, y_f \neq 0 \\ 0, \text{otherwise} \end{cases}
\end{aligned}
\tag{A.31}
$$

where x_f and y_f are the values of feature f, $a_{f,i}(x_f)$ and $a_{f,j}(y_f)$ indicate that the items x and y take the i-th and j-th value of feature f, respectively (see Def. 3). The number of neighbouring values (including the value itself) for feature f, for which two items exhibit non-zero similarity, is denoted by $\mu_f \in \{1, \ldots, q_f\}, \forall f$. The similarity of two items is given by:

$$
\begin{aligned}
\Psi^{ECTE}(\vec{x}, \vec{y}) &= \alpha \sum_f \psi_f^{ECTE}(x_f, y_f) \\
&= \alpha \sum_f \psi_f^{ECTE}(a_{f,i}(x_f), a_{f,j}(y_f)) \\
&= \alpha \sum_{f, x_f \neq 0, y_f \neq 0} \max(0, \frac{\mu_f - |j(y_f) - i(x_f)|}{\mu_f})
\end{aligned}
\tag{A.32}
$$

where $i(x_f)$ and $j(y_f)$ are the indices of values x_f and y_f in A_f (see Fig. A.4c).

As in the case of CTE, each feature is assigned $N_f = N/|F|$ input neurons, and the vector representation of the *input spike pattern* for item x is given in Eq. A.28, with the cosine similarity chosen as a similarity metric. The goal is now to construct vectors $\mathbf{a}_{f,i}$ and $\mathbf{a}_{f,j}$, such that their cosine similarity reproduces the similarity between $a_{f,i}$ and $a_{f,j}$, see Eq. A.31.

The ECTE algorithm which maps values of feature f to binary vectors is given in Alg. 3. Initially, sparse vectors $\mathbf{a}_{f,i}$ are generated as in Def. 6, see Alg. 3 line 6. To control the similarity between vectors, we resort to an *overlap technique*, whereby a partial overlap of vectors associated with different feature values is introduced. The idea is that the resulting overlap should lead to a cosine similarity between two vectors that closely reflects the similarity of two values as given in Eq. A.31 (see Fig. A.4c).

In each non-zero sparse binary vector $\mathbf{a}_{f,i}$ an integer number Δn_f of consecutive elements are replaced. This altered segment of the vector $\mathbf{a}_{f,i}$ is denoted by $\Delta pattern_{f,i} \in \{0,1\}^{\Delta n_f}$.

A segment $\Delta pattern_{f,i}$ consists of μ_f consecutive shorter segments of integer length $\delta n = \frac{\Delta n_f}{\mu_f}$, denoted by $\delta_{f,i,\ell} \in \{0,1\}^{\delta n}$, $\ell = 1, \ldots, \mu_f$. We start by building the initial segment $\Delta pattern_{f,1}$, see Alg. 3 line 10. Each new $\delta_{f,1,\ell}$ segment is built by randomly setting $\eta \leq \delta n$ elements equal to one, whereas the remaining $\delta n - \eta$ elements are set to zero, for $\ell = 1, \ldots, \mu_f$. Then the first Δn_f elements of the initial vector $\mathbf{a}_{f,1}$ are replaced by the obtained segment $\Delta pattern_{f,1}$, see Alg. 3 line 13.

To obtain the next vector, $\mathbf{a}_{f,2}$, we first get $\Delta pattern_{f,2}$ by removing the first δn positions in

$\Delta pattern_{f,1}$, i.e., $\delta_{f,1,1}$, and by inserting a new segment $\delta_{f,2,\mu_f}$ in the last δn positions, see Alg. 3 line 15. We then replace the elements at the positions $\delta n + 1, \delta n + 2, \ldots, \delta n + \Delta n$ of the initial binary vector $\mathbf{a}_{f,2}$ with $\Delta pattern_{f,2}$. As a result, the vectors $\mathbf{a}_{f,1}$ and $\mathbf{a}_{f,2}$ have equal elements at positions $\delta n + 1, \delta n + 2, \ldots, \Delta n_f$, leading to a partial overlap of size $(\mu_f - 1) * \delta n$.

The procedure is iterated $q_f - 1$ times. A new segment $\Delta pattern_{f,i}$ is first obtained from $\Delta pattern_{f,i-1}$ and δ_{f,i,μ_f}. Then the initial vector $\mathbf{a}_{f,i}$ is modified by replacing the elements at the positions $\delta n * (i-1) + 1, \delta n * (i-1) + 2, \ldots, \delta n * (i-1) + \Delta n_f$ with $\Delta pattern_{f,i}$. Therefore, the overlap between vectors associated with consecutive bins has size $(\mu_f - 1) * \delta n$. In general, the overlap between vectors $\mathbf{a}_{f,i}$ and $\mathbf{a}_{f,j}$ is of size $\max(0, \delta n * (\mu_f - |j - i|))$.

Algorithm 3 ECTE

```
Ensure: $\forall i$, construct $\mathbf{a}_{f,i}$
 1: $\Delta pattern_{f,1}$ ← empty array
 2: for $i$ in $1 \ldots q_f$ do
 3:     if $a_{i,f} = 0$ then
 4:         continue
 5:     end if
 6:     $\mathbf{a}_{f,i}$ ← Poisson process with $d_f$
 7:     if i = 1 then
 8:         for $l$ in $1 \ldots \mu_f$ do
 9:             $\delta_{f,1,l}$ ← random_spikes(len=$\delta_n$, num=$\eta$)
10:             $\Delta pattern_{f,1}$ ← concat($\Delta pattern_{f,1}$, $\delta_{f,1,l}$)
11:         end for
12:     end if
13:     $\mathbf{a}_{f,i}[(i-1) * \delta_n : ((i-1) * \delta_n + \Delta n_f)] \leftarrow \Delta pattern_{f,i}$
14:     $\delta_{f,i,\mu_f}$ ← random_spikes(len=$\delta_n$, num=$\eta$)
15:     $\Delta pattern_{f,i+1}$ ← concat($\Delta pattern_{f,i}[\delta_n:]$,$\delta_{f,i,\mu_f}$)
16: end for
```

We observe that the parameters μ_f, η and δ_n need to satisfy the constraints $\forall f$:

$$\delta n(\mu_f + (Q-1)) < n_f \tag{A.33}$$

$$\|\Delta pattern_{f,i}\| = \sqrt{\mu_f \eta} >> \sqrt{d_f} \tag{A.34}$$

$$\eta \leq \delta n \tag{A.35}$$

$$\mu_f \leq q_f \tag{A.36}$$

This encoding method generates vectors $\vec{\mathbf{x}}$ that comply with the desired property in Eq. A.24. From Lemma 1 we know that the inner product, and therefore cosine similarity, vanishes for all pairs of initial vectors $\mathbf{a}_{f,i}$ and $\mathbf{a}_{f,j}$, for $i \neq j$, and for non-zero vectors we have $\|\mathbf{a}_{f,i}\| = \sqrt{d_f}$, as $n \to \infty$. From Eq. A.34 and for large n the cosine similarity of $\mathbf{a}_{f,i}$ and $\mathbf{a}_{f,j}$ is approximately given by the cosine similarity of the overlap between the two vectors:

$$\begin{aligned}
&\forall x, y, \\
\rho(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \approx &\alpha \sum_{f, x_f \neq 0, y_f \neq 0} \rho(\mathbf{a}_{f,i}(x_f), \mathbf{a}_{f,j}(y_f)) \\
\approx &\alpha \sum_{f, x_f \neq 0, y_f \neq 0} \max(0, \frac{\eta(\mu_f - |j(y_f) - i(x_f)|))}{\|\Delta pattern_{f,i}\|^2} \\
= &\Psi^{ECTE}(\vec{x}, \vec{y})
\end{aligned} \tag{A.37}$$

where the first approximation follows from Lemma 2 (see Fig. A.4c). Therefore, the item similarity defined in Eq. (A.31) is approximated for large n by the cosine similarity of the vectors associated with the *input spike patterns* (see Fig. A.4d).

Temporal Encoding of Target Patterns

An output neuron predicts a value zero or one at each time step t, depending on the *input spike pattern*. For each of the $|C|$ input labels a *target spike pattern* is defined, which determines the time instants when the output spikes should occur.

Unlike the generation of the *input spike patterns* where the objective is to approximate the similarity of the input items, in the generation of *target spike patterns* the objective is to obtain orthogonal patterns. To achieve such condition, one could simply choose a one-hot representation for each class, where the number of vector elements is equal to the number of classes, and all elements are zero except the one corresponding to the correct class. One could also assign one output neuron to each class, similarly to what is usually found in ANNs. Here we consider a more general definition, consistent with that of the input encoding function. For each class c, a binary vector $\vec{\mathbf{c}}$ of length $m = OT$ is generated (see Fig. A.4e). For the O output neurons and $|C|$ classes, where $O \geq |C|$, we assign all output neurons of the SNN uniformly among labels, although in general a non-uniform assignment is possible. Without loss of generality, under uniform assignment, the total number of neurons O is assumed to be divisible by the number of labels $|C|$. Therefore, each label will be assigned $O_c = O/|C|$ output neurons.

The vector $\mathbf{c}_i$ associated with label $c_i, 1 \leq i \leq |C|$, is generated as specified in Def. 6 with rate λ_c. The overall vector associated with the target pattern for label i is then given by:

$$\vec{\mathbf{c}} = [0, \ldots, 0, \mathbf{c}_i, 0, \ldots, 0] \tag{A.38}$$

where $(i-1)O_cT$ zeros precede $\mathbf{c}_i$ and $(|C| - i)O_cT$ zeros follow $\mathbf{c}_i$. The *target spike pattern* $\mathrm{X}^{\mathrm{tar}}(c_i)$ represented as a matrix of dimensions $O \times T$ is then obtained by reshaping the vector $\vec{\mathbf{c}}$ similarly as at the input (see Figs. A.4d, A.4e).

A.2.4 Simulation Results

In this section we investigate the performance of the proposed system by simulations that are conducted on two dataset types – images and texts. The MNIST dataset [Deng (2012)] is a well established benchmark in machine learning as well as spiking neural network community. While for MNIST it is possible to obtain high classification accuracy even with binarized input, the Fashion-MNIST [Xiao et al. (2017)] image dataset is more challenging and requires signals with higher precision to be input to the network. The third considered dataset is BBC News [Greene and Cunningham (2006)]. It contains textual articles which should be classified by the topic. We use a permutation invariant version of the three datasets and different fully-connected architectures. Finally, we also demonstrate classification of colored images in 10 and 100 classes (CIFAR10 and CIFAR100 [Krizhevsky et al. (2009)]) for which we consider a spiking convolutional neural network. Through simulations we compare the performance of CTE and ECTE encoding schemes and multiplication-free SNN implementation.

After loading and initial preprocessing of the input dataset, the non-zero features' values are quantized in Q levels (see Def. 3), where bins are determined using the training dataset, such that each bin contains the same number of instances. If many instances have the same value, some bins may be merged, leading to an overall number of quantization levels equal to $q_f \leq Q$. For $Q \geq 2$, far-left and far-right bin edges are set to $-\infty$ and $+\infty$, respectively, to be able to bin large unseen values from validation and test datasets. As a special case, if $Q = 1$, the features are binarized, and all non-zero values belong to the same bin with $q_f = Q = 1$.

The system is set up with hyperparameters $T = 100 * 1ms$, $\nu = 0.8$ and $\beta = 0.07$ (Eqs. A.11, A.12), input rate (see Def. 5) $\lambda_f = 50Hz$ and $d_f = \lambda_f T = 5$, output rate $\lambda_c = 100Hz$ and $d_c = \lambda_c T = 10$, whereas the van Rossum distance metric is computed with $\tau_m = 50$ (Eq. A.14). SNN models are trained with backpropagation through time, using an Adam optimizer [Kingma and Ba (2014)] with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$, exponential learning rate schedule (see Figs. A.5b, A.6b and A.8b) and batch of size 64. The training dataset is shuffled as to avoid the impact of a specific order on the training process. A network is trained using a weighted binary cross-entropy loss (Eq. A.13) and a class is predicted that corresponds to a *target spike pattern* with the smallest van Rossum distance (Eq. A.15, Fig. A.3a). A *target spike pattern* is generated such that the neurons that are activated for the correct class start being active after 40 time steps (see Fig. A.5d), to accommodate for transients of the neuronal membrane potential. We implement regularization with dropout [Srivastava et al. (2014)] and early stopping [Goodfellow et al. (2016)] with patience of 100 epochs.

We compare SNN performance with the performance of a corresponding architecture of a fully-connected feed-forward ANN. For the ANN system, a dataset is loaded and the same preprocessing steps are performed as with SNN. However, in the case of ANN the steps of binning and encoding are ignored and real-valued vectors are sent to the input. The output of the ANN is given by real values that are used to compute the categorical cross-entropy loss. Training of the ANN is done using backpropagation (BP), while optimizer, learning schedule, regularization methods and batch size are identical to what is used with SNN. As a baseline, the comparison is also done with a linear SVM [Cortes and Vapnik (1995)]. The input is exactly the same as for the ANN and L2 regularization is used to avoid overfitting. For MNIST, Fashion-MNIST, CIFAR10 and CIFAR100 datasets, results were obtained on test dataset, while for BBC News, which is a much smaller dataset, we used 10-fold cross-validation. All experiments were performed with 5 trials and mean value and standard deviation are reported.

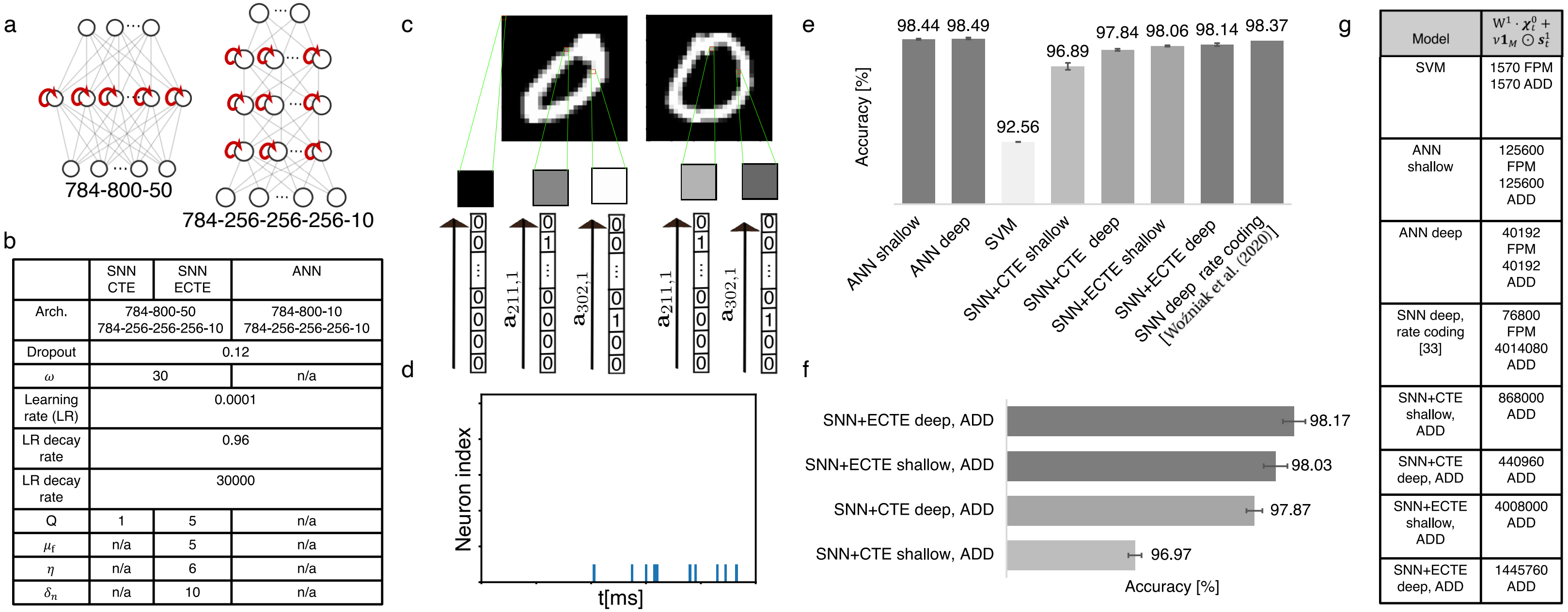
We are also interested in the computational complexity of the proposed SNN system. To focus on the impact of input encoding schemes, we report the number of operations executed in the first hidden layer, i.e. $\mathrm{W}^1 \cdot \boldsymbol{\chi}_t^0$ and $v\mathbf{1}_{M^1} \odot \mathbf{s}_t^1$ (see Eqs. A.11, A.12), for all T time steps. The contribution of the other neuronal operations inside a neuron, such as the computation of activation functions, are ignored as they represent only a small fraction of the two previously stated operations. The vectors $\mathbf{a}_{f,i}$ are stored in a dictionary and in the process of generating *input spike pattern* they are read and concatenated with total complexity of $O(|F|)$. Similarly, assigning existing bins to features' values has complexity of order of magnitude $O(|F|)$. The complexity of initial preprocessing steps will be briefly described for each dataset separately. Similarly, in SVMs we observe the number of operations in matrix-vector multiplication of weights and inputs (the number of additions is taken to be the same as the number of floating point multiplications), whereas other operations are ignored.

MNIST

The MNIST dataset contains 70000 grayscale images of handwritten digits of shape 28 × 28, labeled with one of the ten possible classes. 10000 images are used for testing while the rest 60000 are used for training and validation in ratio 9:1.

As the number of input pixels (features) is relatively large, the number of input neurons N is chosen such that each feature corresponds to exactly one neuron, i.e. $N = |F|$. The performance of the proposed system is tested in two architectures (see Fig. A.6a): a shallow one containing one hidden layer and a large number of neurons (800) and a deeper one containing three hidden layers with 256 neurons each. In a preprocessing step, the pixel values of all images are normalized to the $[0,1]$ range and flattened to 1D arrays. The complexity of these operations is $O(|F|)$. Pixel values are binned and *input spike patterns* are generated using either the CTE or the ECTE algorithm (see Fig. A.6c). At the output of the SNN, each class corresponds to 5 neurons (see Fig. A.6d).

After hyperparameter optimization, we find that the CTE algorithm performs better when the input values are binarized with $Q = 1$, while ECTE yields better performance with higher Q value and number of neighbours μ_f (see Fig. A.5b). For all features f, μ_f was set at 5, except when $q_f < Q$, in which case $\mu_f = q_f$. The chosen values for μ_f, η and δn are satisfying the constrains in Eqs. A.33, A.34, A.35 and A.36. In Fig. A.5e we see that SNN models exhibit performance close to the ANN model and much better than the SVM model. Notably, the ECTE algorithm has a slightly higher accuracy than the CTE algorithm and the deeper architecture



	SNN CTE	SNN ECTE	ANN
Arch.	784-800-50 784-256-256-256-10		784-800-10 784-256-256-256-10
Dropout	0.12		
ω	30		n/a
Learning rate (LR)	0.0001		
LR decay rate	0.96		
LR decay rate	30000		
Q	1	5	n/a
μ_f	n/a	5	n/a
η	n/a	6	n/a
δ_n	n/a	10	n/a

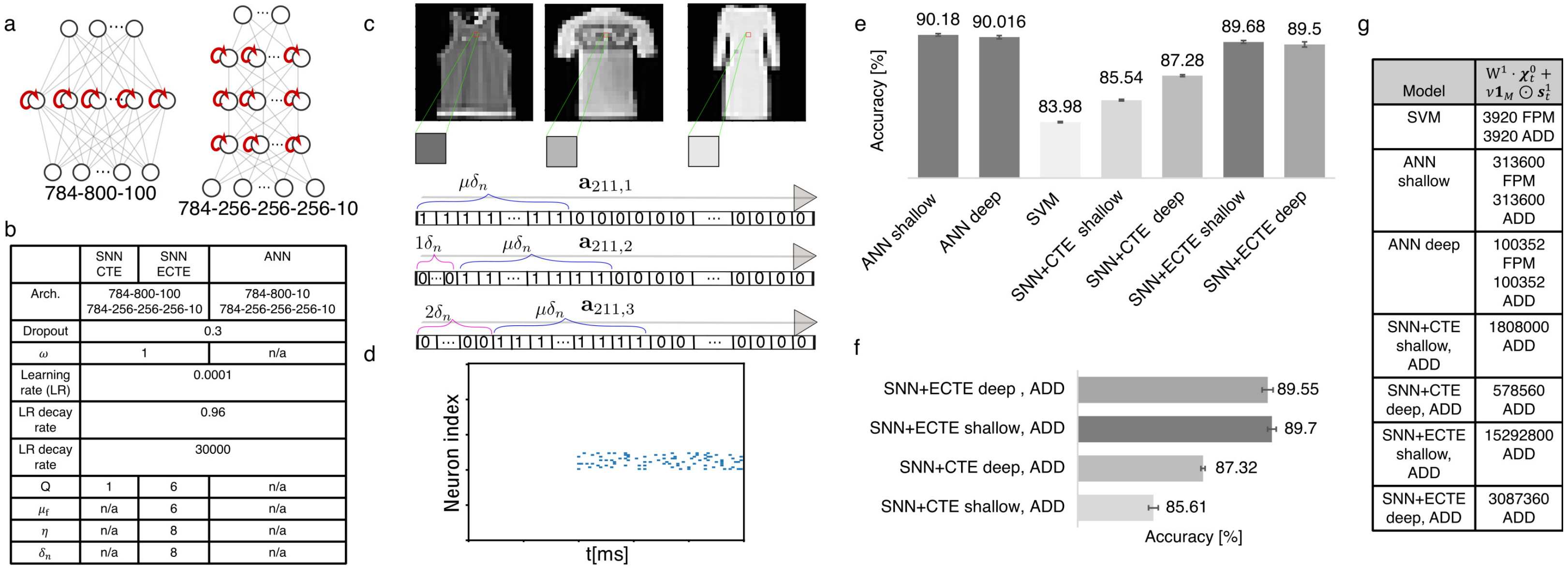
Model	$W^1 \cdot \boldsymbol{\chi}_t^0 + \nu \mathbf{1}_M \odot \boldsymbol{s}_t^1$
SVM	1570 FPM 1570 ADD
ANN shallow	125600 FPM 125600 ADD
ANN deep	40192 FPM 40192 ADD
SNN deep, rate coding [33]	76800 FPM 4014080 ADD
SNN+CTE shallow, ADD	868000 ADD
SNN+CTE deep, ADD	440960 ADD
SNN+ECTE shallow, ADD	4008000 ADD
SNN+ECTE deep, ADD	1445760 ADD

Figure A.5: Setup and results for the MNIST dataset, $|F| = 784$, $T = 100$, $N_f = 1$, $d_f = 5$ a) Two considered SNN architectures: 784-800-50 and 784-256-256-256-10 b) Hyperparameters, μ_f, η and δ_n correspond to ECTE Alg. 3 c) An example of CTE encoding of pixels for two images; a black pixel is encoded as a zero vector, whereas encoding vectors $\mathbf{a}_{f,i}$ of non-zero values are obtained as Poisson processes with parameter d_f; as $Q = 1$, a vector $\mathbf{a}_{f,i}$ is the same for all non-zero values of a feature d) *Target spike pattern* of digit 0; First 40 time steps of all neurons are set to zero e) Accuracy obtained for various systems; "shallow" and "deep" correspond to architectures with one hidden layer and three hidden layers, respectively; f) Accuracy for multiplication-free SNN systems (denoted by 'ADD') g) Number of operations corresponding to first hidden layer for state-of-the-art and multiplication-free SNN systems expressed in additions (ADD) and floating point multiplications (FPM)

performs slightly better than the shallow one. In Fig. A.5f we observe that training and inference with a multiplication-free implementation of the system, doesn't significantly affect the accuracy. Finally, in Fig. A.5g the number of operations is presented for various algorithms. For example, the number of multiplications for the first layer of ANN (and similarly for SVM) is computed as $125600 = 0.2 \cdot 784 \cdot 800$, where we use a factor of 0.2, as on average 80% of input pixels are 0. The number of additions of a first layer of shallow SNN network with CTE encoding is $868000 = 5 \cdot 0.2 \cdot 784 \cdot 800 + 3 \cdot 800 \cdot 100$. The first term corresponds to $\mathrm{W}^1 \cdot \boldsymbol{\chi}_t^0$ and $d_f = 5$ input spikes per neuron and the second term comes from $v\mathbf{1}_{M^1} \odot \mathbf{s}_t^1$ and is computed for 100 time steps where each multiplications is replaced with 3 additions using shift-and-add operations. Similarly, the number of additions of a first layer of deep SNN network with ECTE encoding is $1445760 = 30 \cdot 0.2 \cdot 784 \cdot 800 + 3 \cdot 800 \cdot 100$. The number of input spikes per neuron is dominated by $\Delta pattern_{f,i}$, which has $\mu_f \cdot \eta <= 5 \cdot 6 = 30$ spikes, see Eq. A.34, Alg. 3. For rate coding, the number of operations is calculated for the 5-layer system given in [Woźniak et al. (2020)], where each hidden layer consists of 256 units. The number of multiplications for 300 time steps is as $v\mathbf{1}_{M^1} \odot \mathbf{s}_t^1 = 256 \cdot 300 = 76800$, while the number of additions is $\mathrm{W}^1 \cdot \boldsymbol{\chi}_t^0 = 100 \cdot 0.2 \cdot 784 \cdot 256 = 4014080$ where we assume that all active pixels are white and take the maximal spiking rate of 100.

Fashion-MNIST

The Fashion-MNIST dataset contains 70000 grayscale images of clothes and fashion items of shape 28×28, labeled with one of the ten possible classes, such as 'pullover', 'coat' or 'sandal'. Same as with MNIST, 10000 of images are used for testing while the rest 60000 are used for training and validation in ratio 9:1. N is chosen such that $N = |F|$. Performance is evaluated for a shallow architecture with one hidden layer containing 800 neurons and for a deeper one having three hidden layers with 256 neurons each (see Fig. A.6a). Pixel values of all images are normalized to $[0,1]$, flattened and binned. Using either CTE or ECTE algorithm, bins are encoded into vectors of spikes (see Fig. A.6c) which are then reshaped into *input spike patterns*. Each class is assigned 10 output neurons (see Fig. A.6d).



	SNN CTE	SNN ECTE	ANN
Arch.	784-800-100 784-256-256-256-10		784-800-10 784-256-256-256-10
Dropout	0.3		
ω	1		n/a
Learning rate (LR)	0.0001		
LR decay rate	0.96		
LR decay rate	30000		
Q	1	6	n/a
μ_f	n/a	6	n/a
η	n/a	8	n/a
δ_n	n/a	8	n/a

Model	$\mathrm{W}^1 \cdot \boldsymbol{\chi}_t^0 + \nu \mathbf{1}_M \odot \boldsymbol{s}_t^1$
SVM	3920 FPM 3920 ADD
ANN shallow	313600 FPM 313600 ADD
ANN deep	100352 FPM 100352 ADD
SNN+CTE shallow, ADD	1808000 ADD
SNN+CTE deep, ADD	578560 ADD
SNN+ECTE shallow, ADD	15292800 ADD
SNN+ECTE deep, ADD	3087360 ADD

Figure A.6: Setup and results for the Fashion-MNIST dataset, $|F| = 784$, $T = 100$, $N_f = 1$, $d_f = 5$ a) Two considered SNN architectures: 784-800-50 and 784-256-256-256-10 b) Hyperparameters c) An example of ECTE encoding of pixels for three images; Three consecutive values of 211-th pixel $\mathbf{a}_{211,1}, \mathbf{a}_{211,2}, \mathbf{a}_{211,3}$ are generated by changing corresponding $\mu_f \delta_n$ positions with $\Delta pattern_{211,1}, \Delta pattern_{211,2}, \Delta pattern_{211,3}$ respectively d) *Target spike pattern* of fifth class assigned to label ”Coat” e) Accuracy for various systems f) Accuracy for multiplication-free SNN systems g) Number of operations corresponding to first hidden layer

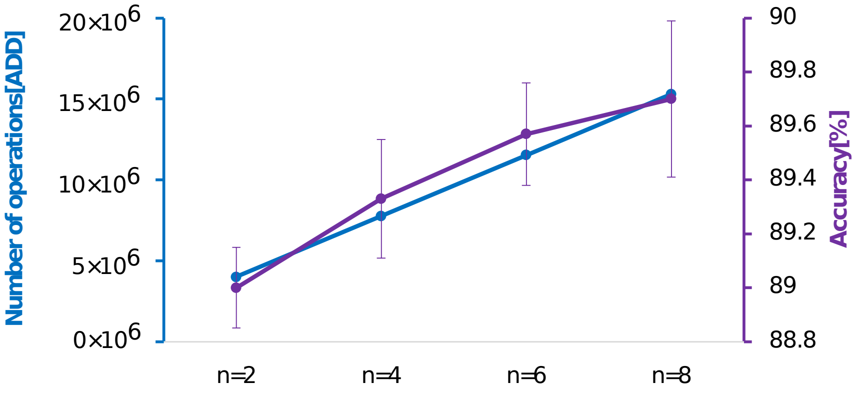


Figure A.7: Accuracy and number of operations for Fashion-MNIST dataset with multiplication-free shallow network and ECTE algorithm; The reduced number of spikes and operations is achieved by decreasing η while other hyperparameters are fixed

The chosen values for Q, μ_f, η and δn are given in Fig. A.5b and satisfy the constrains (see Eqs. A.33, A.34, A.35 and A.36). SNN models show performance close to the ANN model and significantly outperform the SVM model (see Fig. A.6e). Notably, the ECTE algorithm has significantly higher accuracy than the CTE algorithm, which justifies our assertion that Fashion-MNIST is more challenging than MNIST and requires a higher input precision. Interestingly, a deep architecture performs slightly worse than the shallow one, which is also the case for the ANN. This is due to a smaller number of total parameters, once again suggesting that Fashion-MNIST is a more diverse dataset. Multiplication-free implementation of the system doesn't affect the accuracy (see Fig. A.5f). In Fig. A.5g the number of operations of various algorithms is calculated in the same manner as for MNIST, and the number of non-zero pixels is around 50%. Importantly, reducing the number of operations for ECTE algorithm leads to a drop of accuracy, but in a non-linear manner, see Fig. A.7.

BBC News

The BBC News dataset contains 2225 articles labeled with one of five possible topics, such as 'sport' or 'business'. Inputs are used for training and validation in ratio 9:1. Each article is mapped into a real-valued vector where words are features and the tf-idf score [Manning and Schutze (1999); Qaiser and Ali (2018)] denotes their significance in a particular document (see Fig. A.8c). The complexity of these operations is $O(|F|)$. Tf-idf score is computed as:

$$\text{tf\_idf}(\text{word}, \text{doc}, \text{CORP}) = \tag{A.39}$$

$$\text{tf}(\text{word}, \text{doc}) \cdot \text{idf}(\text{word}, \text{CORP}) =$$

$$\frac{freq_{\text{word,doc}}}{\max(freq_{\text{doc}})} \cdot log \frac{|\text{CORP}|}{|\text{CORP}_{\text{word}}|} \tag{A.40}$$

where $freq_{\text{word,doc}}$ denotes the term frequency, i.e., the raw count of a word in a document, $\max(freq_{\text{doc}}) = \max\{freq_{\text{word',doc}}, \text{word'} \in \text{doc}\}$ and $\text{CORP}_{\text{word}} = \{\text{doc} \in \text{CORP} : \text{word} \in \text{doc}\}$. Corpus and ordering of words is determined (fit) from the training data.

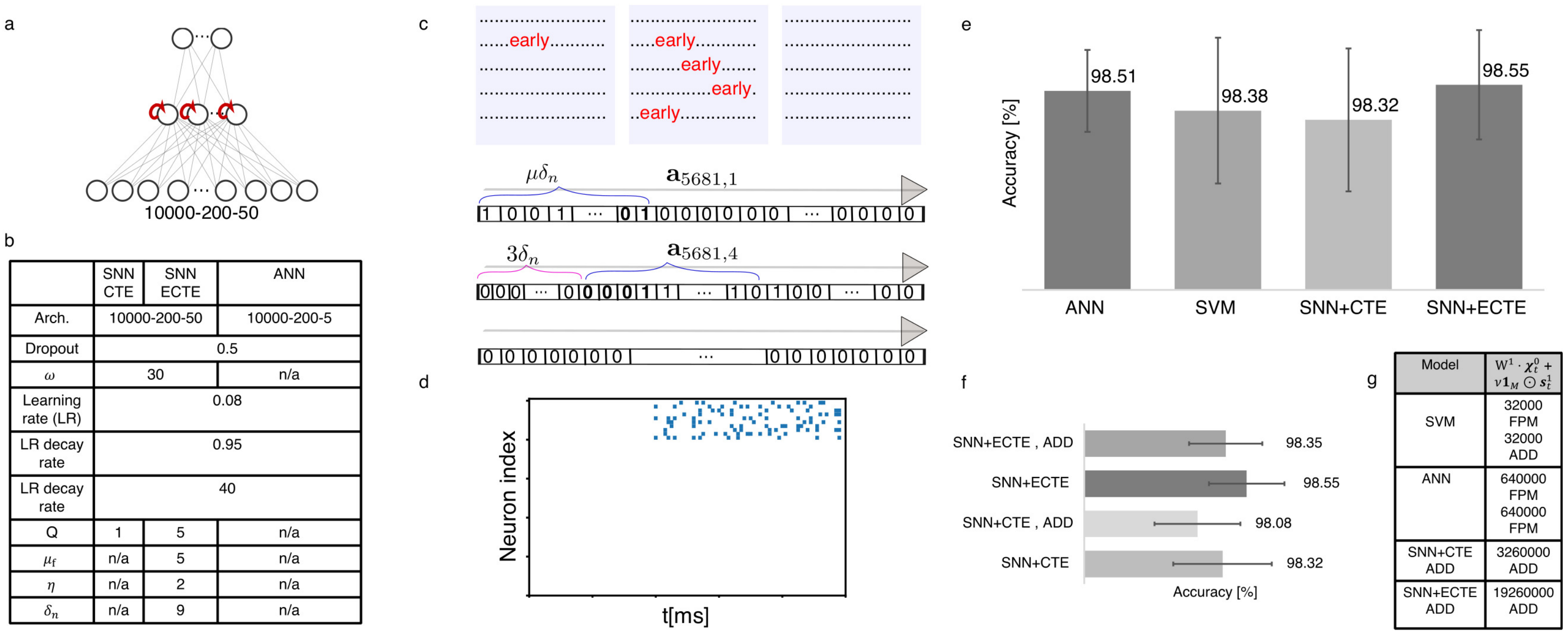
Due to the large number of words in a corpus, the number of input neurons N is chosen such that $N = |F|$. The performance is tested in a shallow architecture with one hidden layer containing 200 units (see Fig. A.8a). Tf-idf values of words are binned and encoded using either CTE or ECTE algorithm (see Fig. A.8c). At the output of SNN, 10 neurons are assigned to each class (see Fig. A.8d).

Fig. A.8b reports the chosen hyperparameters for this dataset. Hyperparameter optimization for Q, μ_f, η and δn yields values that satisfy the constrains in Eqs. A.33, A.34, A.35 and A.36. The performance of SNN system is close to that of the ANN model. Notably, the SVM model with one layer yields a very good accuracy. Unlike image datasets, where classification thrives when pixel values are processed in multiple layers, using deep networks seems not to improve the accuracy in the case of tf-idf values. Notably, the ECTE and CTE algorithms have similar performance (see Fig. A.8e). This suggests that the simple appearance of a particular word in a document brings considerable information, yet the additional tf-idf similarity information conveyed with ECTE further slightly improves the accuracy. In Fig. A.8f we observe that the multiplication-free implementation only slightly reduces the performance of both CTE and ECTE systems. Finally, the number of operations for the different algorithms is computed in the same manner as for the corresponding systems with MNIST and Fashion-MNIST datasets (see Fig. A.8g). The number of non-zero tf-idf values in the input vector is around 32%.

CIFAR10 and CIFAR 100

The CIFAR10 and CIFAR100 datasets contain 60000 colored images of shape $32 \times 32 \times 3$, labeled with one of the 10 and 100 possible classes, respectively. 10000 of images are used for testing while the rest 50000 are used for training. N is chosen such that $N = |F|$. Pixel values of all images are normalized.

We test two different setups. In the first case we use the pretrained ANN weights of the convolutional part of the model based on VGG16 network [Geifman (2018)] to generate a feature vector of size 512. The feature vector is binned and encoded into vectors of spikes using either CTE or ECTE algorithm, and then reshaped into an *input spike pattern*. This input is sent to a fully-connected classifier which is trained, see Fig. A.9a. Each class is assigned 1 and 5 output neurons for CIFAR10 and CIFAR100, respectively. For both CIFAR10 and CIFAR100 the obtained performance shows that there is almost no loss of accuracy of SNN classifier compared to ANN classifier, see Fig. A.9b. For higher efficiency it is also possible to train a spiking convolutional network. To showcase the performance of our approach with convolutional layers, we trained a 4-layer spiking convolutional neural network, see Fig. A.9c. In this case, pixel values of all images are flattened and binned. Using either CTE or ECTE algorithm, bins are encoded into vectors of spikes which are then reshaped into *input spike*



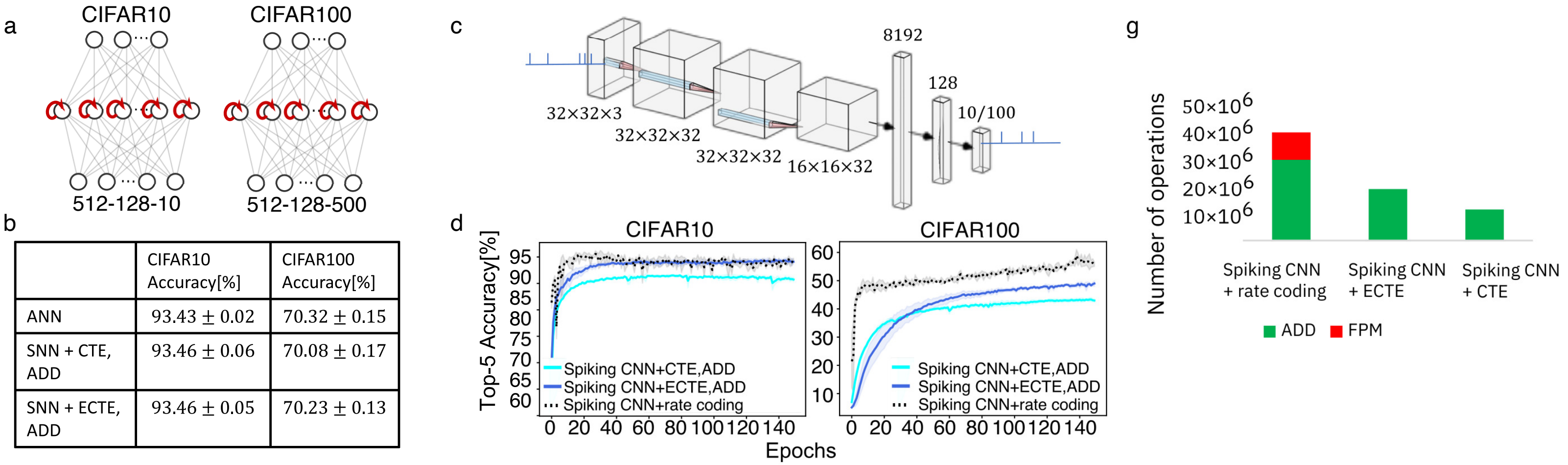
b

	SNN CTE	SNN ECTE	ANN
Arch.	10000-200-50		10000-200-5
Dropout	0.5		
ω	30		n/a
Learning rate (LR)	0.08		
LR decay rate	0.95		
LR decay rate	40		
Q	1	5	n/a
μ_f	n/a	5	n/a
η	n/a	2	n/a
δ_n	n/a	9	n/a

g

Model	$W^1 \cdot \chi_t^0 + \nu \mathbf{1}_M \odot \boldsymbol{s}_t^1$
SVM	32000 FPM 32000 ADD
ANN	640000 FPM 640000 FPM
SNN+CTE ADD	3260000 ADD
SNN+ECTE ADD	19260000 ADD

Figure A.8: Setup and results for the BBC News dataset, $|F| \approx 10000$, $T = 100$, $N_f = 1$, $d_f = 5$ a) SNN Architecture: 10000-200-50; the number of input neurons depends on the number of words in the training data, which is typically around 10000 b) Hyperparameters c) An example of ECTE encoding of word "early" for three articles; word "early" is 5681-th in the fixed ordering; the first document has low tf-idf value, therefore it corresponds to $\mathbf{a}_{5681,1}$, whereas the second document has high tf-idf value and corresponds to $\mathbf{a}_{5681,4}$; the two vectors have small overlap denoted with bold text; the last document doesn't contain the word "early", and is encoded as a zero vector d) *Target spike pattern* of fifth class assigned to label "sport" e) Classification accuracy for various systems f) Accuracy for multiplication-free SNN systems g) Number of operations for the first hidden layer



b

	CIFAR10 Accuracy[%]	CIFAR100 Accuracy[%]
ANN	93.43 ± 0.02	70.32 ± 0.15
SNN + CTE, ADD	93.46 ± 0.06	70.08 ± 0.17
SNN + ECTE, ADD	93.46 ± 0.05	70.23 ± 0.13

Figure A.9: Setup and results for the CIFAR10 and CIFAR100 datasets a) Fully-connected SNN classifier architectures b) Accuracy of CIFAR10 and CIFAR100 datasets obtained with SNN classifiers; The performance is compared with fully-connected ANN classifiers with the same architecture; c) Spiking convolutional neural network trained with both temporal encoding (ECTE and CTE) and rate encoding; d) Top-5 accuracy of the spiking convolutional neural networks g) Number of operations in the first hidden layer of spiking convolutional neural network;

patterns of shape $32 \times 32 \times 3 \times T$. Each class is assigned 1 output neuron. This small network leads to the top-5 classification accuracy of around 95% on CIFAR10 and around 50% on CIFAR100, see Fig. A.9d.

The chosen values of Q, μ_f, η and δn as well as training procedure is exactly the same as for the MNIST dataset, see Fig. A.5b. The dropout value is 0.2, whereas $\omega = 20$. We emphasize that for the convolutional spiking neural network architecture we didn't perform an exhaustive hyperparameter search and the number of training epochs was limited to 150. To make a comparison with rate coding scheme we train the same network with the rate encoded input following the same steps as in [Woźniak et al. (2020)]. In Fig. A.5g the number of operations of various algorithms is calculated in the same manner as for MNIST, and the number of non-zero pixels is estimated at 100%. The rate-coded system requires higher latency and larger number of spikes to achieve high accuracy.

Discussion

Our goal is to devise a low-complexity data classification system, which can be used for various data types. In Fig. A.3a we introduce a computationally efficient SNN model and consider a fully-connected feed-forward architecture, which doesn't assume any prior knowledge on the data type. Figs. A.4, A.5c, A.6c and A.8c depict two novel temporal coding schemes which reduce the number of spikes while retaining important properties, e.g., similarity-preserving input coding and orthogonality of the output patterns. Furthermore, we propose a multiplication-free implementation that is accomplished by removing all multiplications from the model, and replacing them with additions, see Fig. A.3c. Finally, Figs. A.9c and A.9d show that our approach can readily be extended to spiking convolutional neural networks.

In the theoretical study, we introduce similarity metrics (see Eqs. A.22 and A.32) between data items and prove that, for a length of the input patterns $n \to \infty$, our encoding methods preserve similarity in the SNN input. We find that $n = 100$ is sufficient in practice to yield high classification accuracy, whereas higher values of n either bring negligible performance improvements or even reduce the accuracy because of overfitting.

Figs. A.5f, A.6f, A.8f and A.9b show that the proposed system achieves high classification accuracy across various datasets for both the CTE and ECTE schemes with ECTE regularly outperforming CTE. In Figs. A.5g, A.6g, A.8g and A.9e, where the number of operations in the first hidden layer is shown, we see that this higher performance comes at the cost of utilizing a larger number of spikes and operations. Moreover, the multiplication-free system (denoted with ADD) experiences no loss of performance. These methods preserve the similarity of values for individual features, but not across different features. For some datasets, this does not seem to be a limitation, but extending our correlative temporal encoding techniques to also preserve the similarity information across features might represent a promising avenue.

Our method is complementary to the TTFS and rate coding. On one hand, the TTFS technique offers classification with small number of spikes and low latency, however so far this technique has been mainly shown to work in shallow networks [Comşa et al. (2021); Mostafa (2018)]. Furthermore, for each neuronal dynamics, this approach requires a novel mathematical framework for training, whereas in our case BPTT performs the learning for any type of unit. Finally, since our approach utilizes multiple spikes to encode each value, it might be more robust towards jitter and therefore more suitable for hardware implementation. On the other hand, using rate coding and BPTT it is possible to scale SNN models to large number of layers and different neuronal dynamics [Woźniak et al. (2020); Neftci et al. (2019)], however the number of used spikes and latency is high. Our approach can be trained easily with BPTT and scales to larger number of layers, while also being efficient in terms of spikes, number of operations and latency, see Fig A.5g, A.6g, A.8g, A.9e. Moreover, it is well suited for classification tasks, where a continuous stream of input data, e.g., images or text, needs to be processed.

In this work, we are interested in exploring the trade-off between accuracy and resources expressed through the number of parameters, inference delay or computational complexity. The recent results on MNIST dataset show that increasing the amount of resources leads to higher performance. For example, in [Woźniak et al. (2020)], the rate coding utilizes significant number of time steps and spikes which translates to large latency and computational complexity. However, even with a relatively small number of parameters in fully-connected 784-256-256-256-10 architecture and standard leaky integrate-and-fire neuronal dynamics, the authors report a good accuracy of 98.37%. On the contrary, in [Zhang et al. (2019a)], the inference results on the MNIST dataset are obtained with low latency. Nonetheless, even with complex neuronal behaviour which is adapting over time and the large number of parameters in fully-connected 784-1200-1200-10 architecture, the reached accuracy is only 97.2% (see Fig. A.5e). In [Mostafa (2018)], the inference is done with small latency and it requires only one spike per neuron, however the fully-connected 784-800-10 architecture reaches accuracy of only 97.55% and the algorithm does not seem to scale to deeper networks.

When it comes to training SNNs, simpler benchmarks such as MNIST are typically chosen. Here we go a step further and consider two more challenging image datasets as well as a benchmark from the group of natural language processing tasks. One recent work on Fashion-MNIST dataset considers a fully-connected 784-200-200-200-200-200-10 architecture and standard leaky integrate-and-fire neural dynamics [Zhao et al. (2020)]. The reported maximum accuracy of 89.05% is lower than the average accuracy which we obtain for a similar number of parameters. It is worth mentioning that the Fashion-MNIST dataset is often considered with convolutional architectures [Ranjan et al. (2020); She et al. (2019)] which are specialized in image processing. Therefore the results of fully-connected architectures are still 4% lower than the accuracy achieved by the state-of-the-art CNNs. For CIFAR10 and CIFAR100 datasets we consider a fully-connected classifier trained on feature vectors which are obtained from a preprocessing step corresponding to the analogue processing happening in the eye (before the spikes are sent down the optical nerve) [Volobuev et al. (2011)]. Even though the spiking classifier is efficient, the convolutions which are performed in the preprocessing step can lead to large complexity. For an increased energy efficiency some of the later convolutional layers could be transformed into spiking and trained. For the maximal efficiency, we train an end-to-end spiking convolutional neural network, see Figs. A.9c, A.9d. In case of the image datasets, increasing the number of parameters and depth of the network appears to be advantageous in terms of achieved accuracy. In general, employing more complex neuronal dynamics and trying out different shapes of pseudo-derivative during training could lead to interesting insights and increase of the performance.

This work provides a bridge between models using rate coding and the temporal coding with only one spike per neuron. Besides its efficiency, one of the most attractive characteristics of the proposed system is its flexibility. Namely, using well-established frameworks it is easy to train novel architectures and neural dynamics. Importantly, as we can see in Fig. A.7, ECTE method makes it possible to control the trade-off between accuracy and computational complexity through a simple change of η or μ_f hyperparameters in Alg. 3.

The computational complexity of a neuron is often ignored in the SNN literature, even though for large observation intervals it can be dominant and significantly reduce the advantage obtained through spike-based neural communication. In our system, the precision of a multiplication approximation can be readily controlled by adjusting the number of additions (see Eq. A.17). In practice, we find that three shift-and-add operations are sufficient to avoid performance deterioration. For a more detailed analysis, one could compare the computational complexity of the complete system. In this case, the multiplication-free van Rossum distance computation would bring a non-negligible complexity reduction (see Eqs. A.18, A.19). Further improvement of the overall network efficiency could be obtained by regularizing the number of spikes in the hidden layers [Woźniak et al. (2020); Bellec et al. (2018)].

Our complexity calculations aim at providing a basis for investigations on algorithmic trade-offs often posed to SNN systems, without addressing the details of hardware implementation. We remark that addition-based systems exhibit highly desirable properties for the hardware implementation of SNNs [Farsa et al. (2019)], as present hardware solutions suggest that full-precision multiplications are significantly more expensive in terms of energy than additions. In fact, multiplications are estimated to require up to two orders of magnitude more energy per operation than additions, depending on the number of bits and on the adopted VLSI technology [Horowitz (2014); Sze et al. (2019)]. As a final remark, recent works describe hardware-friendly local implementations of the BPTT training algorithm [Bellec et al. (2018); Neftci et al. (2019); Bohnstingl et al. (2022b)], which can potentially be extended to a multiplication-free implementation.

A.2.5 Conclusions and future work

An SNN-based system for classification with low computational complexity was introduced. The complexity was primarily reduced by engaging temporal coding which has been shown to be part of the processing in some areas of the brain and is an efficient way of information encoding. In order to represent the real-valued features in the form of spike trains at the input of the SNN, two novel temporal coding schemes were presented that map such features to correlated *input spike patterns*. Furthermore, we propose the implementation of the system based solely on additions. Various models and configurations are investigated on the data classification task and their accuracy is evaluated on image and textual datasets. Temporal coding schemes together with multiplication-free implementation provide significantly higher efficiency whereas the classification accuracy is at par with that of ANN and linear SVM with full precision multiplications and similar number of parameters. The flexibility of the system makes it easy to combine this work with more complex neuronal dynamics as well as to make trade-offs between resources and performance for each use case. Therefore, our low-complexity and high-accuracy system is a viable choice for neuromorphic hardware.

This work bridges the SNN models using widely adopted yet inefficient rate coding and sparse models built on the premise of temporal coding. Namely, we introduce SNN-based systems where the amount of sparsity and the exact form of the *input spike patterns* are controlled via hyperparameters. For the future it would be interesting to utilize this set up and observe how the number of spikes and the specific form of a temporal coding scheme impact the accuracy for various benchmarks. It is conjectured that inference on certain datasets is impacted stronger by the sparsity. For a concrete hardware implementation this information is important, as the number of spikes and computational complexity translate into energy consumption and area.

The main contribution of this work is provided by two temporal coding schemes, making it possible to reach good performance on benchmark tasks using temporal coding. However, there is room for further improvement in a form of a general theoretical framework that would provide a mapping from analog values to *input spike patterns* without any loss of information. For example, it would be interesting to further generalize the concepts of ECTE which are proven to be effective in an experimental setting.

The proposed SNN-based system reaches its full potential when running on a hardware accelerator, which would fully utilize its sparsity and multiplication-free implementation. The concrete hardware design is a topic of interest and one of the most significant future endeavors. In fact, it is of importance to develop solutions and perform simulations while considering a hardware implementation. Therefore, future work includes testing the impact of quantization on the system performance, as well as implementing a hardware-friendly local realization of the BPTT training algorithm.

A.3 Design of Time-Encoded Spiking Neural Networks in 7nm CMOS Technology

Paper information

Authors: Sandro Widmer, Marcel Kossel, Giovanni Cherubini, Stanisław Woźniak, Pier Andrea Francese, **Ana Stanojevic**, Matthias Brändli, Klaus Frick, Angeliki Pantazi

Abstract In biologically inspired spiking neural networks (SNNs) neurons communicate by short pulses, called spikes. SNNs have the potential to be more power efficient than artificial neural networks (ANNs), thanks to the fewer computational steps required by the spike transmission and processing, as compared to the multiply-and-accumulate (MAC) operations with wide bitvectors usually adopted in ANNs. We present the design of two types of SNNs with integrate-and-fire dynamics and single-spike per neuron operation, where neural communication is based on synchronous time-to-first-spike (sTTFS) and time-to-first-spike (TTFS) encoding schemes. In the considered time-encoded SNNs, the information is carried by the timing of the spikes with respect to a reference time. In 7nm CMOS technology both designs are synthesized as VHDL-based random-logic-macros (RLMs) and compared to an equivalent ANN design in terms of power consumption, latency and silicon area, using the Iris data set for inference. A cost function expressed as a product of energy consumption and silicon area is introduced to compare the three network designs. With respect to this cost function, it turns out that the SNN-TTFS implemented for the considered classification task outperforms the ANN used as baseline model.

Keywords Spiking neural networks (SNNs), artificial neural networks (ANNs), synchronous time-to-first-spike (sTTFS) encoding, time-to-first-spike (TTFS) encoding

Publication [Widmer et al. (2023)] (https://ieeexplore.ieee.org/document/10129114).

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Moshe Abeles. Local cortical circuits: An electrophysiological study. *Springer*, 1982.

Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. *Advances in neural information processing systems*, 32, 2019.

Henrik Alle and Jorg RP Geiger. Combined analog and action potential coding in hippocampal mossy fibers. *Science*, 311(5765):1290–1293, 2006.

Yuki M. Asano, Christian Rupprecht, Andrew Zisserman, and Andrea Vedaldi. Pass: An imagenet replacement for self-supervised pretraining without humans. *NeurIPS Track on Datasets and Benchmarks*, 2021.

David Attwell and Simon B Laughlin. An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow & Metabolism*, 21(10):1133–1145, 2001.

Craig H Bailey, Maurizio Giustetto, Yan-You Huang, Robert D Hawkins, and Eric R Kandel. Is heterosynaptic modulation essential for stabilizing hebbian plasiticity and memory. *Nature Reviews Neuroscience*, 1(1):11–20, 2000.

Susan M Barnett and Stephen J Ceci. When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological bulletin*, 128(4):612, 2002.

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.

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.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

Guo-qiang Bi and Mu-ming Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, 24(1):139–166, 2001.

Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

Elie L Bienenstock, Leon N Cooper, and Paul W Munro. Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2(1):32–48, 1982.

Thomas Bohnstingl, Ayush Garg, Stanisław Woźniak, George Saon, Evangelos Eleftheriou, and Angeliki Pantazi. Speech recognition using biologically-inspired neural networks. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6992–6996, 2022a. doi: 10.1109/ICASSP43922.2022.9747499.

Thomas Bohnstingl, Stanisław Woźniak, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022b.

Sander M Bohte. Error-backpropagation in networks of fractionally predictive spiking neurons. In *International Conference on Artificial Neural Networks*, pages 60–68. Springer, 2011.

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.

Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 159–172. IEEE, 2021.

Irem Boybat, Manuel Le Gallo, S. R. Nandakumar, Timoleon Moraitis, Thomas Parnell, Tomas Tuma, Bipin Rajendran, Yusuf Leblebici, Abu Sebastian, and Evangelos Eleftheriou. Neuromorphic computing with multi-memristive synapses. *Nature Communications*, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

T. Bu, W. Fang, J. Ding, P. Dai, Z. Yu, and T. Huang. Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. *ICLR*, 2022.

Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863*, 2019.

Geoffrey W Burr, Robert M Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, et al. Neuromorphic computing using non-volatile memory. *Advances in Physics: X*, 2(1):89–124, 2017.

Zhiqiang Cao, Long Cheng, Chao Zhou, Nong Gu, Xu Wang, and Min Tan. Spiking neural network-based target tracking control for autonomous mobile robots. *Neural Computing and Applications*, 26(8):1839–1847, 2015.

Giovanni Cherubini, Jens Jelitto, and Vinodh Venkatesan. Cognitive storage for big data. *Computer*, 49:43–51, September 2016.

Giovanni Cherubini, Ana Stanojevic, and Abu Sebastian. Correlative time coding method for spiking neural networks, 2022. US Patent 11,403,514.

Iulia-Maria Comşa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function: Learning with backpropagation. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

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. *Proceedings of the National Academy of Sciences*, 119(4):e2109194119, 2022.

Sanjoy Dasgupta, Charles F. Stevens, and Saket Navlakha. A neural algorithm for a fundamental computing problem. *Science*, 358(6364):793–796, 2017. doi: 10.1126/science.aam9868.

Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: solving the credit assignment problem without a backward pass. In *International Conference on Machine Learning*, pages 4937–4955. PMLR, 2022.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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.

Katharina Eichler, Feng Li, Ashok Litwin-Kumar, Youngser Park, Ingrid Andrade, Casey M. Schneider-Mizell, Timo Saumweber, Annina Huser, Claire Eschbach, Bertram Gerber, Richard D. Fetter, James W. Truman, Carey E. Priebe, L. F. Abbott, Andreas S. Thum, Marta Zlatic, and Albert Cardona. The complete connectome of a learning and memory centre in an insect brain. *Nature*, 2017. doi: 10.1038/nature23455.

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

Edris Z Farsa, Arash Ahmadi, Mohammad A Maleki, Morteza Gholami, and Hima N Rad. A low-cost high-speed neuromorphic hardware based on spiking neural network. *IEEE Transactions on Circuits and Systems—II: Express Briefs*, 66(9):1582–1586, 2019.

Shuo Feng, Haowei Sun, Xintao Yan, Haojie Zhu, Zhengxia Zou, Shengyin Shen, and Henry X Liu. Dense reinforcement learning for safety validation of autonomous vehicles. *Nature*, 615(7953):620–627, 2023.

Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85, 2016.

Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44 (1):154–180, 2020.

Avishek Garain, Arpan Basu, Fabio Giampaolo, Juan D Velasquez, and Ram Sarkar. Detection of covid-19 from ct scan images: A spiking neural network-based approach. *Neural Computing and Applications*, 33(19):12591–12604, 2021.

Brian Gardner, Ioana Sporea, and André Grüning. Learning spatiotemporally encoded pattern transformations in structured spiking neural networks. *Neural computation*, 27(12):2548–2586, 2015.

Yonatan Geifman. Github, 2018. URL https://github.com/geifmany/cifar-vgg.

Robert Geirhos, Carlos RM Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge, and Felix A Wichmann. Generalisation in humans and deep neural networks. *Advances in neural information processing systems*, 31, 2018.

W. Gerstner. Spiking neurons. In W. Maass and C. M. Bishop, editors, *Pulsed Neural Networks*, chapter 1, pages 3–53. MIT-Press, 1998.

W. Gerstner and W. K. Kistler. *Spiking Neuron Models: single neurons, populations, plasticity*. Cambridge University Press, Cambridge UK, 2002.

W. Gerstner, W.M. Kistler, R. Naud, and L. Paninski. *Neuronal Dynamics. From single neurons to networks and cognition.* Cambridge Univ. Press, 2014.

Alireza Ghahari and John D Enderle. A neuron-based time-optimal controller of horizontal saccadic eye movements. *International journal of neural systems*, 24(06):1450017, 2014.

Samanwoy Ghosh-Dastidar and Hojjat Adeli. Third generation neural networks: Spiking neural networks. In *Advances in Computational Intelligence*, pages 167–178. Springer, 2009.

Tim Gollisch and Markus Meister. Rapid neural coding in the retina with relative spike latencies. *Science*, 319(5866):1108–1111, 2008.

J. Göltz, L. Kriener, A. Baumbach, S. Billaudelle, O. Breitweiser, B. Cramer, D. Dold, A.F. Kungl, W. Senn, J. Schemmel, K. Meier, and M.A. Petrovici. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature Machine Intelligence*, 3:823–835, 2021.

Julian Göltz, Andreas Baumbach, Sebastian Billaudelle, AF Kungl, Oliver Breitwieser, Karlheinz Meier, Johannes Schemmel, Laura Kriener, and Mihai A Petrovici. Fast and deep neuromorphic learning with first-spike coding. In *Proceedings of the neuro-inspired computational elements workshop*, pages 1–3, 2020.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep Learning.* MIT Press, Cambridge Mass., 2016.

Michael Graupner and Nicolas Brunel. Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location. *Proceedings of the National Academy of Sciences*, 109(10):3991–3996, 2012.

Derek Greene and Pádraig Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proc. 23rd International Conference on Machine learning (ICML'06)*, pages 377–384. ACM Press, 2006.

Benedikt Grothe. How the barn owl computes auditory space. *Trends in neurosciences*, 41(3): 115–117, 2018.

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.

Donald O Hebb. The first stage of perception: growth of the assembly. *The Organization of Behavior*, 4(60):78–60, 1949.

S. Hochreiter, Y. Bengion, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S.C. Kremer and J.F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

SG Hu, GC Qiao, TP Chen, Qi Yu, Y Liu, and LM Rong. Quantized stdp-based online-learning spiking neural network. *Neural Computing and Applications*, 33(19):12317–12332, 2021.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

C.P. Hung, G. Kreiman, T. Poggio, and J.J. DiCarlo. Fast readout of object identity from macaque inferior temporal cortex. *Science*, 310:863 – 866, 2005.

Eric Hunsberger and Chris Eliasmith. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141*, 2016.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

B. Illing, J. Ventura, G. Bellec, and W. Gerstner. Local plasticity rules can learn deep representations using self-supervised contrastive predictions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 30365–30379. Curran Associates, Inc., 2021.

Bernd Illing, Wulfram Gerstner, and Johanni Brea. Biologically plausible deep learning—but how far can we go with shallow networks? *Neural Networks*, 118:90–101, 2019.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.

Mikolaj Jankowski, Deniz Gündüz, and Krystian Mikolajczyk. Joint device-edge inference over wireless links with pruning. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2020.

Ziheng Jiang, Tianqi Chen, and Mu Li. Efficient deep learning inference on edge devices. *ACM SysML*, 2018.

Cristian Jimenez-Romero and Jeffrey Johnson. Spikinglab: modelling agents controlled by spiking neural networks in netlogo. *Neural Computing and Applications*, 28(1):755–764, 2017.

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

Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, and Sarah Mack. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

Saeed Reza Kheradpisheh and Timothée Masquelier. Temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, 30(06): 2050027, 2020.

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.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Helga Kolb, Eduardo Fernandez, and Ralph Nelson. Webvision: the organization of the retina and visual system [internet]. *University of Utah Health Sciences Center*, 1995.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Toronto, ON, Canada*, 2009.

M Fabiana Kubke, Dino P Massoglia, and Catherine E Carr. Developmental changes underlying the formation of the specialized time coding circuits in barn owls (tyto alba). *Journal of Neuroscience*, 22(17):7671–7679, 2002.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Peter Lennie. The cost of cortical computation. *Current biology*, 13(6):493–497, 2003.

Bin Li, Yuki Todo, and Zheng Tang. Artificial visual system for orientation detection based on hubel–wiesel model. *Brain Sciences*, 12(4):470, 2022.

Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2): 566–576, 2008. doi: 10.1109/JSSC.2007.914337.

Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Yan Wang, Yongjian Wu, Feiyue Huang, and Chia-Wen Lin. Rotated binary neural network. *Advances in neural information processing systems*, 33:7474–7485, 2020.

Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Fei Chao, Mingliang Xu, Chia-Wen Lin, and Ling Shao. Siman: Sign-to-magnitude network binarization. *arXiv preprint arXiv:2102.07981*, 2021.

Junxiu Liu, Dong Jiang, Yuling Luo, Senhui Qiu, and Yongchuang Huang. Minimally buffered deflection router for spiking neural network hardware implementations. *Neural Computing and Applications*, 33(18):11753–11764, 2021.

Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737, 2018.

Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *European conference on computer vision*, pages 143–159. Springer, 2020.

Yihao Luo, Haibo Shen, Xiang Cao, Tianjiang Wang, Qi Feng, and Zehan Tan. Conversion of siamese networks to spiking neural networks for energy-efficient object tracking. *Neural Computing and Applications*, 34(12):9967–9982, 2022.

Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

CN Marimuthu, P Thangaraj, and Aswathy Ramesan. Low power shift and add multiplier design. *arXiv preprint arXiv:1006.1179*, 2010.

Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. A history of spike-timing-dependent plasticity. *Frontiers in synaptic neuroscience*, 3:4, 2011.

Timothée Masquelier and Simon J Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS computational biology*, 3(2):e31, 2007.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Alexander Meulemans, Matilde Tristany Farinha, Javier Garcia Ordonez, Pau Vilimelis Aceituno, João Sacramento, and Benjamin F Grewe. Credit assignment in neural networks through deep feedback control. *Advances in Neural Information Processing Systems*, 34:4674–4687, 2021.

Maryam Mirsadeghi, Majid Shalchian, Saeed Reza Kheradpisheh, and Timothée Masquelier. Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing*, 427:131–140, 2021.

H. Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3227–3235, 2018. doi: 10.1109/TNNLS.2017.2726060.

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.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15:267–273, 1982.

John O'Keefe and Michael L Recce. Phase relationship between hippocampal place units and the eeg theta rhythm. *Hippocampus*, 3(3):317–330, 1993.

L. M. Optican and B. J. Richmond. Temporal encoding of two-dimensional patterns by single units in primate inferior temporal cortex. 3. Information theoretic analysis. *J. Neurophysiol.*, 57:162–178, 1987.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

Seongsik Park, Seijoon Kim, Byunggook Na, and Sungroh Yoon. T2fsnn: Deep spiking neural networks with time-to-first-spike coding. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7):18–28, 2022.

Jean-Pascal Pfister and Wulfram Gerstner. Triplets of spikes in a model of spike timing-dependent plasticity. *Journal of Neuroscience*, 26(38):9673–9682, 2006.

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.

J.W. Pillow, J. Shlens, L. Paninski, A. Sher, A. M. Litke, E. J. Chichilnisky, and E.P. Simoncelli. Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature*, 454:995–999, 2008.

Filip Ponulak and Andrzej Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71(4):409–433, 2011.

Shahzad Qaiser and Ramsha Ali. Text mining: Use of tf-idf to examine the relevance of words to documents. *International Journal of Computer Applications*, 181, 07 2018. doi: 10.5120/ijca2018917395.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

Joshua Arul Kumar Ranjan, Titus Sigamani, and Janet Barnabas. A novel and efficient classifier using spiking neural network. *The Journal of Supercomputing*, 76(9):6545–6560, 2020.

F. Rieke, D. Warland, R. de Ruyter van Steveninck, and W. Bialek. *Spikes - Exploring the neural code*. MIT Press, Cambridge, MA, 1996.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Bodo Rueckauer and Shih-Chii Liu. Conversion of analog to spiking neural networks using sparse temporal coding. In *2018 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2018.

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.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Chaitanya Ryali, John Hopfield, Leopold Grinberg, and Dmitry Krotov. Bio-inspired hashing for unsupervised similarity search. In *International Conference on Machine Learning*, pages 8295–8306. PMLR, 2020.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

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.

Abu Sebastian, Manuel Le Gallo, Geoffrey W Burr, Sangbum Kim, Matthew BrightSky, and Evangelos Eleftheriou. Tutorial: Brain-inspired computing using phase-change memory devices. *Journal of Applied Physics*, 124(11):111101, 2018.

Xueyuan She, Yun Long, and Saibal Mukhopadhyay. Fast and low-precision learning in gpu-accelerated spiking neural network. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 450–455. IEEE, 2019.

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.

Gordon Simons and N. L. Johnson. On the convergence of binomial to poisson distributions. *Ann. Math. Statist.*, 42(5):1735–1736, 10 1971. doi: 10.1214/aoms/1177693172.

M. Sorbaro, Q. Liu, M. Bortone, and S. Sheik. Optimizing the energy consumption of spiking neural networks for neuromorphic applications. *Front. Neurosci.*, 14:662, 2020.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Ana Stanojevic, Giovanni Cherubini, Timoleon Moraitis, and Abu Sebastian. File classification based on spiking neural networks. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

Ana Stanojevic, Evangelos Eleftheriou, Giovanni Cherubini, Stanisław Woźniak, Angeliki Pantazi, and Wulfram Gerstner. Approximating relu networks by single-spike computation. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 1901–1905. IEEE, 2022a.

Ana Stanojevic, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini, Angeliki Pantazi, and Wulfram Gerstner. An exact mapping from relu networks to spiking neural networks. *arXiv preprint arXiv:2212.12522*, 2022b.

Ana Stanojevic, Giovanni Cherubini, Stanisław Woźniak, and Evangelos Eleftheriou. Time-encoded multiplication-free spiking neural networks: application to data classification tasks. *Neural Computing and Applications*, 35(9):7017–7033, 2023a.

Ana Stanojevic, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini, Angeliki Pantazi, and Wulfram Gerstner. Are training trajectories of deep single-spike and deep relu network equivalent? *arXiv preprint arXiv:2306.08744*, 2023b.

C. Stockl and W. Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nat. Mach. Intell.*, 3:230–238, 2021.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13693–13696, 2020.

S.C. Surace, J.-P. Pfister, W. Gerstner, and J. Brea. On the choice of metric in gradient-based theories of brain function. *PLoS Comput. Biol.*, 16:e1007640,, 2020.

David Sussillo and LF Abbott. Random walk initialization for training very deep feedforward networks. *arXiv preprint arXiv:1412.6558*, 2014.

Vivienne Sze, Tien ju Yang, Yu hsin Chen, and Joel Emer. Efficient processing of deep neural networks: from algorithms to hardware architectures. In *NeurIPS*, page 138, 2019.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019.

S. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *Nature*, 381:520–522, 1996.

S. Thorpe, A. Delorme, and R. Van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14:715–725, 2001.

Mesut Toğaçar, Burhan Ergen, and Zafer Cömert. Detection of weather images by using spiking neural networks of deep learning models. *Neural Computing and Applications*, 33 (11):6147–6159, 2021.

Gina Turrigiano. Homeostatic synaptic plasticity: local and global mechanisms for stabilizing neuronal function. *Cold Spring Harbor perspectives in biology*, 4(1):a005736, 2012.

Mark CW van Rossum. A novel spike distance. *Neural computation*, 13(4):751–763, 2001.

Vinodh Venkatesan, Taras Lehinevych, Giovanni Cherubini, Andrii Glybovets, and Mark Lantz. Graph-based data relevance estimation for large storage systems. In *2018 IEEE International Congress on Big Data (BigData Congress)*, pages 232–236. IEEE, 2018.

Andrey N Volobuev, Eugeny S Petrov, et al. Analog-to-digital conversion of information in the retina. *Natural Science*, 3(01):53, 2011.

Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020.

Yikai Wang, Yi Yang, Fuchun Sun, and Anbang Yao. Sub-bit neural networks: Learning to compress and accelerate binary neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5360–5369, 2021.

Sandro Widmer, Marcel Kossel, Giovanni Cherubini, Stanisław Woźniak, Pier Andrea Francese, Ana Stanojevic, Matthias Brändli, Klaus Frick, and Angeliki Pantazi. Design of time-encoded spiking neural networks in 7nm cmos technology. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 1–1, 2023. doi: 10.1109/TCSII.2023.3277784.

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.

Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems*, 4:795–813, 2022.

Timo C Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11:12829, 2021.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zihan Xu, Mingbao Lin, Jianzhuang Liu, Jie Chen, Ling Shao, Yue Gao, Yonghong Tian, and Rongrong Ji. Recu: Reviving the dead weights in binary neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5198–5208, 2021.

Adam Yala, Constance Lehman, Tal Schuster, Tally Portnoi, and Regina Barzilay. A deep learning mammography-based model for improved breast cancer risk prediction. *Radiology*, 292(1):60–66, 2019.

D.L.K. Yamins and J.J. DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nat. Neurosci.*, 19:356–365, 2016.

Zhanglu Yan, Jun Zhou, and Weng-Fai Wong. Near lossless transfer learning for spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10577–10584, 2021.

K. Yang, J.H. Yau, J. Deng, and O. Russakovsky. A study of face obfuscation in imagenet. *Proc. 39th Intern. Conf. Machine Learning PMLR*, 162(1):25313–25330, 2022.

Gang Yu, Kai Sun, Chao Xu, Xing-Hua Shi, Chong Wu, Ting Xie, Run-Qi Meng, Xiang-He Meng, Kuan-Song Wang, Hong-Mei Xiao, et al. Accurate recognition of colorectal cancer with semi-supervised deep learning on pathological images. *Nature communications*, 12(1):6311, 2021.

Davide Zambrano, Roeland Nusselder, H Steven Scholte, and Sander M Bohté. Sparse computation in adaptive spiking neural networks. *Frontiers in neuroscience*, 12:987, 2019.

F. Zenke and T.P. Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Computation*, page 899–925, 2021.

Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

Friedemann Zenke, Everton J Agnes, and Wulfram Gerstner. Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks. *Nature communications*, 6(1):1–13, 2015.

Anguo Zhang, Hongjun Zhou, Xiumin Li, and Wei Zhu. Fast and robust learning in spiking feed-forward neural networks based on intrinsic plasticity mechanism. *Neurocomputing*, 365:102–112, 2019a.

Lei Zhang, Shengyuan Zhou, Tian Zhi, Zidong Du, and Yunji Chen. Tdsnn: From deep neural networks to deep spike neural networks with temporal-coding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1319–1326, 2019b.

Malu Zhang, Jiadong Wang, Jibin Wu, Ammar Belatreche, Burin Amornpaisannon, Zhixuan Zhang, Venkata Pavan Kumar Miriyala, Hong Qu, Yansong Chua, Trevor E Carlson, et al. Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(5):1947–1958, 2021.

Dongcheng Zhao, Yi Zeng, Tielin Zhang, Mengting Shi, and Feifei Zhao. Glsnn: A multi-layer spiking neural network based on global feedback alignment and local stdp plasticity. *Frontiers in Computational Neuroscience*, 14, 2020.

Bolei Zhou. Github, 2018. URL https://github.com/zhoubolei.

Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1452–1464, 2017.

PERSONAL INFORMATION

Ana Stanojevic

📍 Wasserschöpfi 49, 8055 Zurich, Switzerland

☎ +41786832054

✉ anastanojevic93@gmail.com

github.com/ana-stanojevic

Date of birth 26/06/1993 | Nationality Serbian | Swiss residence permit C

PROFESSIONAL EXPERIENCE

(Oct. 2019 - Oct. 2023)

Predoctoral researcher, machine learning

IBM Research, Zurich (https://www.zurich.ibm.com/)

- Topic: spiking neural networks - learning algorithms, input encoding schemes, neural dynamics, applications of spiking neural networks in big data

(Feb. 2019 - Aug. 2019)

Researcher, machine learning (Internship)

IBM Research, Zurich (https://www.zurich.ibm.com/)

- Master's thesis topic: File Classification Based on Spiking Neural Networks
  Supervisors: Prof. Dr. Wulfram Gerstner, Dr. Giovanni Cherubini, Dr. Abu Sebastian
- Technologies: **TensorFlow, Python;**

(Aug. 2018 - Feb. 2019)

Software engineer (Internship)

Google, Zurich (https://www.google.com/about/)

- Topic: Google Assistant product, Improving the relevance of a given answer in case when user issues multiple dependent queries
- Technologies: **Python, C++**;

(Jun. 2017 - Feb. 2018)

Data scientist (Junior)

Deeption, Lausanne (http://deeption.com//)

- Topic: Analysing public opinion on social media
- Technologies: **Python (data science libraries), Twitter API, Stanford Parser, FastText, SIF embedding, Gephi, Elastic Search, JavaScript;**

EDUCATION

(Feb. 2020 - Nov. 2023)

PhD student of computer and communication sciences

Laboratory of computational neuroscience (LCN), Supervisor: Prof. Wulfram Gerstner

École Polytechnique Fédérale de Lausanne (EPFL), School of Computer and Communication Sciences, Lausanne

- Subjects - Networks out of control, Deep Learning in Artificial and Biological Neuronal Networks, Spatio-Temporal Modelling in Biology, Mobile Health and Activity Monitoring;
- **Semester project:** "Time-to-First-Spike Coding for Spiking Neural Networks", LCN;
- **Semester project:** "Efficient Hardware Implementation of Gaussian Proc. Modelling", LIA

(Sept. 2016 - Sept. 2019)

Master of Science in Communication Systems

Specialization: Data analytics

École Polytechnique Fédérale de Lausanne (EPFL), School of Computer and Communication Sciences, Lausanne

- Subjects - Pattern classification and machine learning; Applied data analysis; Analytic algorithms; Cryptography and security; Computer vision; Distributed information systems, Data visualisation, Information theory and coding, A network tour of data science, Entrepreneurship laboratory (e-lab), Deep learning, Lab in data science, Artificial neural networks, Statistical signal and data processing, Industrial automation;
- **Semester project:** "Data analytics of MOOC (massive open online course) data", CHILI lab;

(Sept. 2012 - Aug. 2016)

Bachelor of Science in Electrical and Computer Engineering

School of Electrical Engineering, Department of Computer and Software Engineering, Belgrade

- Subjects - Algorithms and Data Structures; Databases; System Programming; Computer networks; Compilers; Software Design; VLSI Computer Systems;
- **Bachelor's thesis:** Development of ML tool for restoring diacritical characters in text.

TEACHING EXPERIENCE

(Jan. 2021 - Jan. 2023)

Teaching Assistant

École Polytechnique Fédérale de Lausanne (EPFL), School of Computer and Communication Sciences, Lausanne

- **Course:** Artificial Neural Networks

(Feb. 2017 - Jul. 2018) MOOC Teaching Assistant (“Smart Cities - Management of Smart Urban Infrastructures”)

IGLUS - Innovative Governance of Large Urban Systems, Lausanne (http://iglus.org/)

- Daily discussions with international participants about **Smart Cities and Internet of Things.**

TECHNICAL SKILLS

- **Programming languages/Libraries:** C, C++, Java, Python, Matlab, JavaScript, Bash, Keras, PyTorch, sklearn, pandas, Tensorflow;
- **Big Data:** Apache Spark, Apache Hadoop HDFS, Apache Kafka**;**
- **Misc:** Mathematica, LaTex, SQL, git, Docker**;**

AWARDS AND DIPLOMAS

- **EPFL Excellence Fellowship,** Lausanne**,** 2016-2018
- **Dositeja award**, Serbian Government Fund for Young Talents, Scholarships for the best students of the final years, Serbia, 2015/2016
- **Scholarship for exceptionally talented high school and university students,** Ministry of Education, Science and Technological Development, Serbia, 2009-2015
- **Certificate for successful completion of International Summer School for Young Physicist**, Perimeter Institute, Canada, 2010

SELECTED PUBLICATIONS AND PATENTS

- **Giovanni Cherubini, Ana Stanojevic, Abu Sebastian, “Correlative time coding method for spiking neural networks”, US Patent App. 15/931,514, 2021**
- **Stanojevic, Ana, et al. "Approximating ReLU Networks by Single-Spike Computation”**, ***2022 IEEE International Conference on Image Processing (ICIP)*, 2022**
  We show that an SNN with time-to-first-spike (TTFS) encoding and a specific piecewise linear postsynaptic potential approximates a standard ReLU network. Adapting batch normalisation paradigm during learning yields ANN-like performance of our 2-layer SNN on image classification task (MNIST).
- **Stanojevic, Ana, et al. "Time Encoding Schemes in Spiking Neural Networks with no Multiplications: Data Classification Application”**, **2022, Neural Computing and Applications journal**
  We leverage surrogate gradient for SNN training, and propose two temporal encoding schemes which translate the similarity of the analog input into a correlation of the corresponding input spike patterns. Using our multiplication-free SNN system it is possible to obtain a good performance on image classification tasks (MNIST and CIFAR10).
- **Stanojevic, Ana, et al. "An Exact Mapping from ReLU Networks to Spiking Neural Networks”**, ***2023,* Neural Networks, *under review***
  We find that there exist a neural dynamics and a set of parameter constraints which guarantee an approximation-free mapping from ReLU network to an SNN with TTFS encoding. A pretrained deep ReLU network can be replaced with deep SNN without any performance loss on large-scale image classification tasks (CIFAR100 and PLACES365).
- **Stanojevic, Ana, et al. "Are training trajectories of deep single-spike and deep ReLU network equivalent?”**, ***2023,* NeurIPS2023, *under review***
  We discover that the reason for unsuccessful training of deep SNNs with TTFS encoding is vanishing-and-exploding gradient problem. A particular neural dynamics and a set of parameter constraints solves the problem and yields the same learning trajectories as ReLU network on large image classification tasks (CIFAR100 and PLACES365).

LANGUAGES

	LISTENING	READING	SPEAKING	WRITING
English	C2	C2	C1	C1
	Certification - TOEFL iBT: 102 points (120 maximum), Sept. 2015			
German	B1	A2	B1	A2
	fide-Test, Jan. 2023			
French	A1	A1	A1	A1
	Attestation – EPFL Centre de langues – Modules Intensifs – Niveau A1, Aug. 2016			

VOLUNTEERING EXPERIENCE

(Sept. 2019 - Jun. 2021) IT Trainer

Powercoders (A coding academy for refugees), Zurich (https://powercoders.org/)