OPTIMIZING RATE-BASED SPIKING NEURAL NETWORKS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2022

Student id: 10382958

Chen Li
Department of Computer Science, School of Engineering

Contents

Word Count: 31821

List of Tables

List of Figures

Abstract

OPTIMIZING RATE-BASED SPIKING NEURAL NETWORKS
Chen Li
A thesis submitted to The University of Manchester
for the degree of Doctor of Philosophy, 2022

A notable trend in recent years is the transition of the prevalent deep learning algorithms to edge devices, and a primary concern is unsustainable energy dissipation. Deep SNN, benefiting from their event-based nature and efficient information communication by spikes, can serve as a competitive candidate for achieving a more power-efficient computing paradigm. A practical way to train an SNN is to first train an ANN and then convert it into a rate-coded SNN, a method called ANN-to-SNN conversion. This method enables building functional SNNs at a low cost and validating various optimization strategies in SNNs.

Based on ANN-to-SNN conversion, this thesis explores the rationale behind SNNs and the optimization of SNNs from various aspects. First, it clarifies the fundamental question of why to use SNNs. Few advantages of SNNs compared with conventional ANN have been found up to now. The presented results show that SNNs can render better robustness to noisy synaptic weights. This research paves the way for applying memristors, a cutting-edge component with intrinsic noise, to spike-based in-memory computing. Second, it focuses on retaining the biological plausibility of state-of-the-art SNNs. In the presented study, the neuronal dynamics of the standard integrate-and-fire model are analyzed, and the difficulty of weight-bias imbalance when using this model is relieved. Better accuracy is achieved than the state-of-the-art SNNs. Third, the accuracy-latency trade-off, one of the essential challenges in rate-coded SNNs, is alleviated in the presented study. It elaborates on the role of noise in fast SNNs and the necessity of information compression in achieving low-latency SNNs. The SNNs optimized by this approach achieved an accuracy of 70.18% in 8 times steps on ImageNet. Finally, SNNs need to be deployed to neuromorphic hardware or neuromorphic chips for real-world applications. An SNN deployment on SpiNNaker is described in this thesis, featuring high accuracy (98.63% on MNIST), structural plasticity, and low firing rates.

Declaration

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

Copyright

Glossary

ANN Artificial neural network (ANN) is a deep neural network model, the dominant model in deep learning. 6–9, 14–17, 33, 36–38, 40, 42, 43, 50, 51, 53, 54, 57–60, 62–64, 67, 68, 71, 73, 75, 77–85, 87–89, 92, 93, 95–97, 99–105, 107, 109–121, 123

ANN-to-SNN conversion ANN-to-SNN conversion is a method to build a spiking neural network by converting an artificial neural network. 6–9, 16, 38–42, 51, 53, 57–60, 64, 77, 78, 81–83, 87, 89, 90, 92, 93, 95, 96, 98–101, 103, 104, 109–114, 118–120, 122–124

API Application programming interface (API) is a way for two or more computer programs to communicate with each other. 15, 50, 101

CPU Central processing unit (CPU) is the electronic circuitry that executes instructions comprising a computer program. 14, 50, 52, 55, 92

GPU Graphics processing unit (GPU) is a specialized electronic circuit designed to accelerate certain computations such as the creation of images. 14, 15, 50–52, 55, 92

SNN Spiking neural network (SNN) is a neural network model made up of spiking neurons. 6–9, 14–20, 22, 24, 28, 31–36, 38, 40–45, 48–89, 92–104, 106–124

Acknowledgements

For the chance provided to demonstrate my passion on a PhD, I express the gratitude to my supervisor Prof. Steve Furber. He gave the maximum freedom as well as appropriate support during his supervision to me. His outlook and sharp view exemplified how a capable researcher looks like.

I thank my colleagues Edward Jones, Luca Peres, Adam Perrett, Petrut Bogdan, Gengting Liu, Oliver Rhodes, Jim Garside, John Woods, Andrew Rowley, Minmin Jiang, Sara Summerton, Tingting Mu.

I would like to express my sincere thanks to my parents Xuejun Li and Lifang Qiao. My special thanks to Luhui Liu for her emotional supports and companion.

Chapter 1

Introduction

1.1 Overview of context

Neuromorphic computing, whose origins can be traced back to the 1980s, is meeting new challenges and opportunities. In its initial definition, neuromorphic computing is using very-large-scale integration (VLSI) circuits to emulate biological neurons and neural systems formed by these neurons. The rationale for implementing spiking neurons on hardware is twofold: promoting a better understanding of the brain by implementing intelligence in silicon, and exploring bio-inspired computing during this implementation of silicon neurons. The scope of neuromorphic computing has gradually expanded, and silicon neurons are implemented on more architectures. In this century, various projects have received generous funding to build large-scale neuromorphic machines as well as to validate neuromorphic chips.

At the same time, the third wave of artificial intelligence has surged from 2006 [HOT06], featuring the adoption of deep learning techniques. The success of AlexNet in 2012 [KSH12] and the new ImageNet records that surpass the human level in 2015 [RDS$^+$15, HZRS15, HZRS16] are two influential landmarks in deep learning. Nevertheless, these deep learning algorithms are run on CPUs, GPUs, and neural network accelerators, instead of on neuromorphic hardware.

A pioneering paper was published in 2015 [DNB$^+$15], which demonstrates the conversion of ANNs, a dominant technique in deep learning, to SNNs, a model that can be simulated on neuromorphic hardware. This research has had a long-term impact on neuromorphic computing and inspired many following studies. On the one hand, by converting an ANN to an SNN and then implementing it on neuromorphic hardware, neuromorphic hardware can implicitly run deep learning algorithms, the current dominant AI method. On the other hand, considering the reported similarity between ANNs and SNNs, some deep learning tools that are initially designed for ANNs can be utilized to simulate SNNs.

A key challenge to be met by neuromorphic machines is the effective and efficient deployment of SNN algorithms. “Effective” refers to the lossless deployment of a SNN from GPUs to neuromorphic hardware. The accuracy loss can come from the different neuronal dynamics of SNNs simulated on GPUs and SNNs implemented on neuromorphic hardware, as well as from different APIs and hardware architectures. “Efficient” refers to the power-efficient SNN implementation in real-time. Algorithm optimization is proven to be a significant method to optimize for these two aspects. For instance, an SNN can be optimized to adapt to the constraints of neuromorphic hardware so bringing better accuracy and a lower burden on memory and computing units.

Simulating SNNs on deep learning tools offers many beneficial effects such as accelerating SNN prototyping and validation, and driving faster progress on algorithm development and optimization. By simulating SNNs on deep learning tools, many pivotal problems such as why to use SNNs instead of ANNs, how to build noise-robust SNNs, and how to build fast SNNs can be studied. The knowledge that appears during this process can be transmitted on neuromorphic hardware to implement functional SNNs on practical tasks.

In summary, SNNs are promising to bring deep learning techniques to neuromorphic computing, which provides new opportunities to implement functional spike-based algorithms on neuromorphic hardware for real-world tasks. There are some challenges associated with these opportunities along this new journey, which have drawn increased scholarly attention.

1.2 Objectives and contributions

As described in the section above, the new trend, leveraging the techniques and tools from deep learning to improve SNN performance and expand SNN scope, provides new opportunities and challenges. To capture these opportunities and overcome the challenges met by current SNNs, this thesis presents several original research that contributes to the development of SNNs. The presented research in this thesis covers several key aspects of SNNs, including deploying SNN on neuromorphic hardware (Section 3.5) and cutting-edge materials (Section 3.4), the advantages of using SNNs over traditional ANNs (Section 3.1 to Section 3.3), the biological plausibility of SNNs (Section 4), and the inference latency of SNNs (Section 5). The works covered in this thesis are not independent but closely related to each other. By combing them together, a complete workflow to apply SNNs to real-world applications with high biological plausibility and performance is presented: from why to use SNNs to how to improve their performance; from SNN simulation and optimization on simulators, to SNN deployment on current neuromorphic hardware and neuromorphic hardware that contains noisy but energy-efficient components.

Central to this thesis is the optimization of rate-based deep spiking neural networks. The studies presented in this thesis close the gap between ANNs and SNNs with regard to inference performance, and reveal the merits of SNNs such as its noise robustness and biological plausibility. The main contributions are:

- **A quantization framework for fast SNNs** (Chapter 5): The performance of SNNs, and specifically their inference accuracy, has improved significantly over recent years, which can be attributed to the bridge built by pioneer researchers to carry knowledge from ANNs over to SNNs, a method called ANN-to-SNN conversion. In this study, a second bridge from ANNs to SNNs is built, with the primary goal to reduce the inference latency of SNNs. This study provides a comprehensive quantization framework for fast SNNs, and investigates the challenges of information compression and noise suppression. The results show this approach achieves state-of-the-art accuracy and latency (70.18% in 8 time steps on ImageNet) compared with other low-latency SNNs built by ANN-to-SNN conversion. In summary, the accuracy-latency trade-off in SNNs is alleviated by this study.

- **An SNN normalization method to balance network performance and biological fidelity** (Chapter 4): Except for the well-known accuracy-latency trade-off in SNNs, a trade-off between network performance and biological plausibility exists in the current SNN research. The recent trend in SNN research is trading biological plausibility for better performance. This study demonstrates that state-of-the-art SNN accuracy can also be achieved with more biologically-plausible neuronal models and input coding schemes. The main challenge solved in this study is the modeling of bias and batch normalization, two key elements in ANNs, by the standard spiking neurons, which enables converting more powerful ANN model to build SNNs with higher accuracy. The SNNs achieved 99.71% accuracy on MNIST and 93.6% accuracy on CIFAR-10.

- **Noisy weights** (Chapter 3): After the proposal of the first method to build functional SNNs, there is considerable scholarly attention attached to the advantages of SNNs compared to ANNs. This study suggests that SNNs are more robust to noisy synaptic weights than ANNs. This research contributes to our understanding of the dynamics of spiking neurons, specifically, their characteristics on noise robustness. This research may also be of value to memristor-based neuromorphic computing where the memristor, a noisy cutting-edge device, is adopted as non-volatile memory to store weights efficiently.

- **SNN implementation on SpiNNaker** (Chapter 3): Ultimately, an SNN algorithm will be deployed to neuromorphic hardware for real-world applications. This study reports the deployment of a deep SNN algorithm on SpiNNaker, a digital neuromorphic hardware. The reported results outperform other SNN implementations on SpiNNaker (0.43% higher accuracy on MNIST).

1.3 Publications

There are several published peer-reviewed papers to support the contributions of this thesis.

- **C. Li**, S. Furber, **Towards Biologically-Plausible Neuron Models and Firing Rates in High-Performance Deep Spiking Neural Networks**, International Conference on Neuromorphic Systems (ICONS 2021). This paper [LF21] provides a method to model bias and batch normalization by the standard integrate-and-fire model in SNNs, which obviously improves the SNN accuracy on MNIST and CIFAR-10. Meanwhile, biological plausibility is retained. The contents of this paper are presented in Chapter 4.

- **C. Li**, R. Chen, C. Moutafis, S. Furber, **Robustness to noisy synaptic weights in spiking neural networks**, International Joint Conference on Neural Networks (IJCNN 2020). This study [LCMF20] shows that SNNs are more robust to Gaussian noise in synaptic weights than ANNs under some conditions. This finding will enhance our understanding of the merits of SNNs compared with ANNs. Also, these results imply the possibility of using high-performance cutting-edge materials with intrinsic noise as an information storage medium in SNNs. This study is shown in Chapter 3.

- R. Chen, **C. Li**, C. Moutafis, S. Furber, **Nanoscale room-temperature multilayer skyrmionic synapse for deep spiking neural Networks**, Physical Review Applied. This research [CLL+20] proposes a nanoscale skyrmionic synapse composed of magnetic multilayers that enables room-temperature device operations tailored for optimal synaptic resolution. A method to embed such multilayer skyrmionic synapses in spiking neural networks is provided, and results show that an accuracy of 98.61% is achieved on MNIST. This research illustrates that the proposed skyrmionic synapse can be a potential candidate for future energy-efficient neuromorphic edge computing. Chapter 3 includes a part of the contents of this study.
- **C. Li**, L. Ma, S. Furber, **Quantization Framework for Fast Spiking Neural Networks**, Frontiers in Neuroscience. This study elaborates a comprehensive quantization framework to build SNNs with ultra-low latency. The significance of information compression and noise suppression is emphasized in this paper, with an in-depth analysis and corresponding practical solutions. State-of-the-art SNN latency is achieved on ImageNet. This study is introduced in Chapter 5.

1.4 Thesis structure

The rest of this thesis is composed of the following chapters:

Chapter 2 introduces the fundamental elements in both classical spiking neural networks and deep spiking neural networks, and the overview of neuromorphic hardware and deep learning.

Chapter 3 investigates the robustness of noisy weights in spiking and non-spiking neural networks, as well as the architecture design and optimizations of an SNN built with skyrmionic synapses. Also, a demonstration of an SNN on SpiNNaker is provided.

Chapter 4 discusses the imbalance between weights and biases in SNNs when applying the standard integrate-and-fire model, and provides a feasible solution to overcome this challenge.

Chapter 5 proposes a quantization framework for achieving low-latency SNNs, and highlights the role of information compression and noise suppression.

Chapter 6 summarizes the research presented in this thesis and suggests future directions.

Chapter 2

Background

This chapter provides background information on deep spiking neural networks. In the first two sections, classical spiking neural networks and neuromorphic hardware are introduced. After the surge of deep learning in 2012, spiking neural networks gradually go deeper and transition to deep spiking neural networks. Section 2.3 gives a brief overview of deep learning. The related elements and techniques about deep spiking neural networks are introduced in Section 2.4. Section 2.5 summarizes the contents covered in this chapter.

2.1 Classical spiking neural networks

SNNs are neural networks that contain structured biologically-inspired spiking neuronal models which are connected by synapses. In SNNs, the information is processed in spiking neurons, carried by "all-or-none" spikes, and propagated through synapses. These information processing systems are researched intensively to explore the reasons behind the astonishing spatiotemporal data processing ability in the brain, and to mimic these brain merits and duplicate them in other platforms by reverse engineering. Spiking neurons are sparsely activated and their firing time is usually not synchronized. These features allow SNNs to be efficiently implemented on neuromorphic machines [FGTP14, MAAI$^+$14, DSL$^+$18, SBG$^+$10] which are often event-based, asynchronous, and highly-parallel.

SNNs are divided into two categories in this thesis—classical SNNs and deep SNNs—according to whether they follow the deep learning paradigm and apply deep learning techniques such as gradient descent and backpropagation. This section covers the contents of classical SNNs, with an emphasis on their computational properties

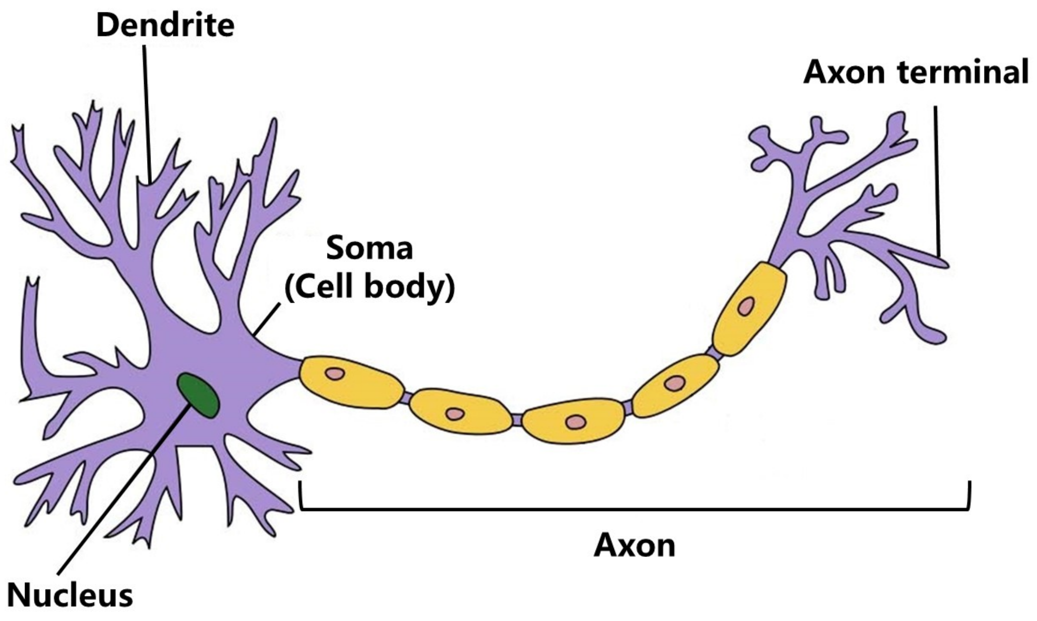

Figure 2.1: The anatomy of a neuron. The figure is modified from [Tow91].

and biological plausibility. The over-detailed biological foundations of SNNs, such as some experimental neuroscience observations and some computational neuroscience conclusions, are out of the scope of this section.

2.1.1 Anatomy of a neuron

A neuron (also known as a nerve cell) comprises three parts: dendrites, a soma, and an axon. Communication between neurons relies on synapses. As shown in Figure 2.1, dendrites are tree-shaped structures that receive information from other neurons through synapses. The soma, also known as the cell body, is the core of a neuron that maintains the normal function of the neuron, e.g. by producing proteins in the nucleus of the soma. The axon is a tail-like structure that sends signals to other neurons. Locating at the end of the axon are axon terminals, and they contain neurotransmitters that are crucial for signal transmission from this neuron to another. Neurons do not touch each other directly, but are connected by synapses. Specifically, a synapse connects an axon terminal of a neuron to a dendrite of the next neuron. Neurons are extensively linked to other neurons in this way, enabling information to be carried to different neurons.

The carrier of information is spike. When spikes are generated and propagated through the axon to the axon terminals, neurotransmitters are released into the synapses and accepted by the dendrites of other neurons. During this process, the electrical signals

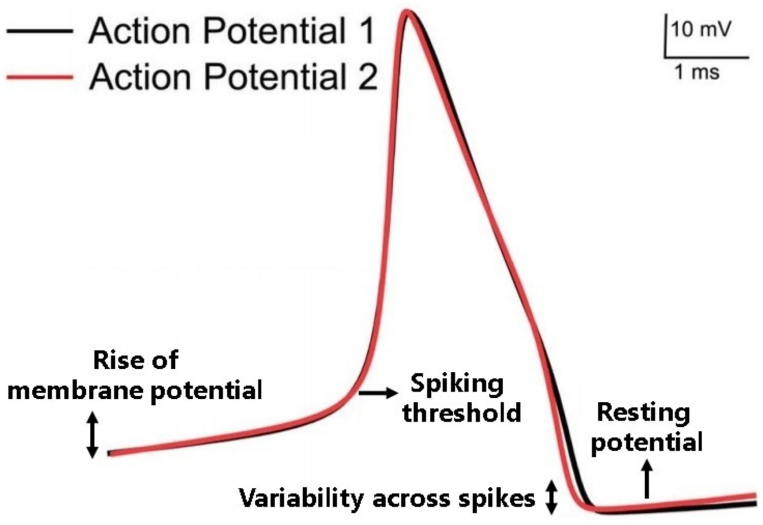

Figure 2.2: The waveform of two action potentials with similar height and width. The Figure is modified from [SS14].

(spikes) in the neurons are converted to chemical signals (carried by neurotransmitters) and then converted to electrical signals (input currents) in the other neurons.

2.1.2 Spikes

A spike is an electrical signal occurring in a neuron, and it is also known as an action potential or a nerve impulse. When a spike is generated in a spiking neuron, the membrane potential of this neuron rapidly rises and falls, forming a pulse-shaped electrical signal traveling along the axon of this neuron. The waveforms of spikes are identical, and follow a stereotypical voltage change and time duration as shown in Figure 2.2. After a spike is emitted, this neuron enters a refractory period with a typical time length of several milliseconds during which spikes are harder to generate.

Whether a spike is generated is determined by the values of its membrane potential and spiking threshold. The membrane potential is the difference between the inside and outside electrical potentials in a neuron, represented by voltage with the units of millivolts. The spiking threshold is usually modeled by a voltage value that provides the explicit criterion to determine whether a spike will be generated, though the biological details are more complicated. A more detailed description of the spike generation process is given along with an introduction to the IF model in the following section.

2.1.3 Neuronal models

There are a variety of neuronal models proposed to capture the dynamics of spiking neurons in nervous systems. This section introduces five typical models which are widely supported in neuromorphic hardware and SNN simulators. These models are all single-compartment point neurons which means they ignore the complicated dendrite dynamics and the physical shape of spiking neurons. The computational complexity and biological plausibility of these five models are compared in Figure 2.3.

- **Integrate-and-fire model (IF model)**. This is also known as the non-leaky integrate-and-fire model, and it is one of the earliest and simplest spiking neuronal models [Abb99]. The neuronal dynamics of the IF model can be summarised as two stages: voltage integration and spike firing. At the voltage integration stage, the input current is integrated into the membrane potential of a spiking neuron, and causes the membrane potential (which is usually measured by voltage. Typical values of membrane potential are in the range from -70 mV to -40 mV.) to increase or decrease; this voltage will keep stable if no more input electrical signals are integrated. Once the voltage inside a spiking neuron surpasses a threshold, it enters the second stage, spike firing. At this stage, a spike will be generated, and the membrane potential will drop down to the resting potential (e.g. -70 mV) and wait for the next integration.

- **Leaky integrate-and-fire model (LIF model)**. The LIF model is more biologically plausible than the standard IF model, as it takes the imperfection of membrane potential into consideration. Due to this imperfection, the integrated voltage in a spiking neuron will slowly leak out over time, which is modeled by the leak mechanism in the LIF model. The leak rate in an LIF neuron is controlled by a time constant whose typical value is several milliseconds. The LIF model is the default spiking model in many neuromorphic machines and SNN simulators. More detailed neuronal dynamics of the LIF model are illustrated by equations in the following section.

- **Other variants of Integrate-and-fire model**. In addition to the LIF model, there are other variants of the standard IF model, e.g. the adaptive integrate-and-fire model which is useful in online learning in SNNs. There are one or more parameters in the adaptive IF model that can adapt to the input current. This neuronal model can fit and predict the data from experimental neuroscience more

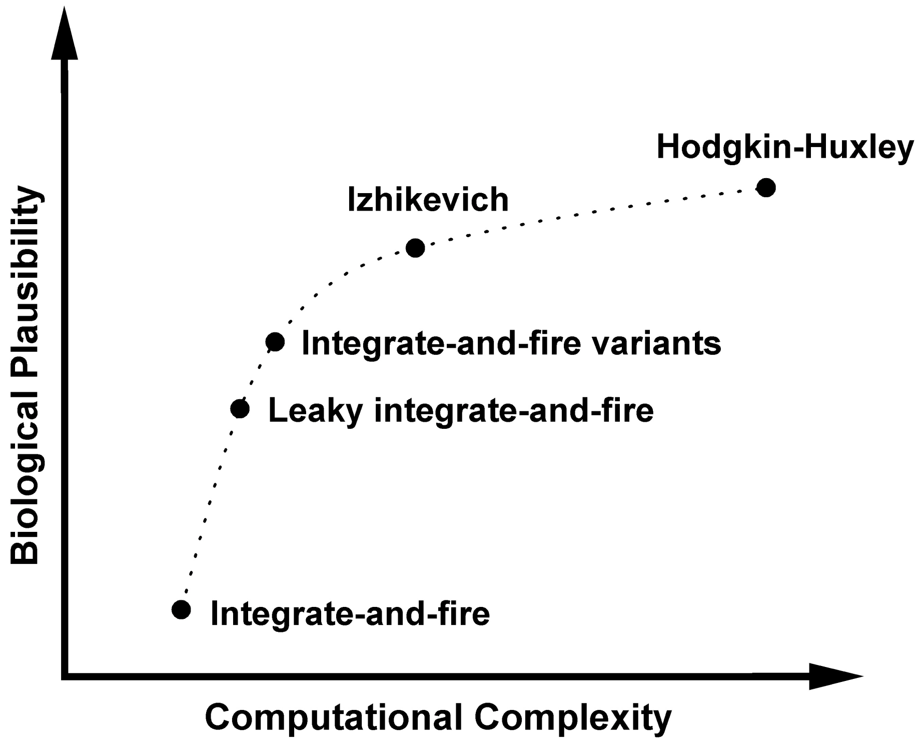

Figure 2.3: Neuronal models with different computational complexity and biological plausibility. The figure is modified from [Izh04]. The detailed position of different neuronal models is disputable.

precisely, so it is more biologically plausible than the standard IF model and, potentially, the LIF model.

- **Hodgkin-Huxley (HH) model**. The Hodgkin-Huxley neuronal model [HH52] was proposed in 1952. It consists of several nonlinear differential equations to model the impact of different ion channels to the membrane potential of spiking neurons. In contrast, there is only one differential equation in the IF model and the LIF model.

- **Izhikevich model**. The Izhikevich spiking model [Izh03] was proposed in 2003, with the rationale to reconcile computational efficiency and biological plausibility in spiking neuronal modeling. As shown in Figure 2.3, the biological plausibility of the Izhikevich model is higher than the IF models and the IF model variants, while its computational complexity is lower than the HH model. [Izh04] shows that the Izhikevich model is in a “sweet spot”: it has a similar computational efficiency to the IF model and the same number of biologically-plausible features as the HH model.

2.1.4 Neuronal dynamics of the LIF model

The LIF model is widely supported in neuromorphic machines and SNN simulators as a basic neuronal model. This section describes the detailed neuronal dynamics of the LIF model using equations. It starts by deriving the two main equations that control the subthreshold and supra-threshold dynamics of the LIF model. Some useful conclusions, including how the membrane potential is charged, instantly charged, and leaked, can be gotten from the subthreshold equation. These conclusions enable the adoption of the simplified computation of LIF neurons during SNN simulations, and facilitate the understanding of the dynamics of LIF neurons. Combining the subthreshold equation and the supra-threshold equation, the output spike rates of an LIF neuron under different input can be calculated. This input-output response is crucial for rate-coded SNNs, and is elaborated in this section as well.

LIF Equations

LIF model consists of three mechanisms: integration, leak, and firing. The first two mechanisms, also known as the subthreshold dynamics of an LIF neuron, can be modeled by an RC circuit as shown in Figure 2.4 . This RC circuit has a resistor with the resistance R, connected to a power supply whose voltage is u_{rest}, in parallel with a capacitor whose capacitance is C. The input to this circuit is the current $I(t)$. According to Kirchhoff's current law,

$$I(t) = I_R + I_C, \tag{2.1}$$

in which

$$I_R = \frac{u(t) - u_{rest}}{R}, \tag{2.2}$$

and

$$I_C = \frac{\mathrm{d}q}{\mathrm{d}t} = C\frac{\mathrm{d}u}{\mathrm{d}t}. \tag{2.3}$$

Combining Equation 2.2 and 2.3 with Equation 2.1 gives

$$I(t) = \frac{u(t) - u_{rest}}{R} + C\frac{\mathrm{d}u}{\mathrm{d}t}. \tag{2.4}$$

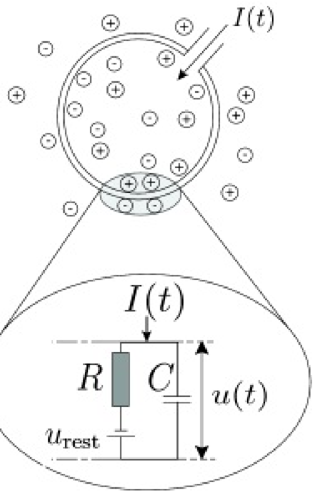

Figure 2.4: An LIF neuron and its RC circuit model. A neuron enclosed by the cell membrane receives an input current $I(t)$ which causes the change of its membrane potential. The cell membrane acts like a capacitor in parallel with a resistor which is in line with a battery of potential u_{rest} (zoomed inset). Figure is from [GKNP14].

Multiplying both sides of this equation by R and introducing the time constant of the membrane potential $\tau_m = RC$, this becomes:

$$\tau_m \frac{\mathrm{d}u}{\mathrm{d}t} = -[u(t) - u_{rest}] + RI(t), \tag{2.5}$$

which is the standard equation defining the subthreshold dynamics of an LIF neuron.

The spike generating mechanism (also known as the supra-threshold dynamics) of an LIF neuron is controlled by its threshold $u_{threshold}$. Equation 2.6 describes how the output spike train is calculated. Once $u(t) > u_{threshold}$, a spike $z(t)$ is generated at time t and the membrane potential $u(t)$ is reset to resting potential u_{rest}; on the other hand, no spike will be generated when $u(t) \leq u_{threshold}$.

$$z(t) = \begin{cases} 1 \textit{ and } u(t) = u_{rest}, & \textit{when } u(t) > u_{threshold}, \\ 0, & \textit{when } u(t) \leq u_{threshold}. \end{cases} \tag{2.6}$$

Note that the values “1” and “1” of $z(t)$ represent the all-or-none property of a spike rather than describing its explicit form and shape.

The leak of the pre-charged voltage

Considering an LIF neuron with an initial membrane potential $u(t_0)$ under the constraint $u_{rest} < u(t_0) \leq u_{threshold}$ and zero input current $I(t)$,

$$u(t_0) = u_{rest} + \Delta u, \tag{2.7}$$

and

$$I(t) = 0, \ for \ t \geq t_0. \tag{2.8}$$

The solution of Equation 2.5 with these initial conditions is

$$u(t) - u_{rest} = \Delta u e^{\frac{t-t_0}{\tau_m}}, \ for \ t \geq t_0. \tag{2.9}$$

It suggests that in the absence of input current, the membrane potential of an LIF neuron decays exponentially with time, and the decay rate is characterized by the time constant τ_m. When $t = \infty$, Equation 2.9 becomes

$$u(t) - u_{rest} = 0. \tag{2.10}$$

This equation shows that after infinite time t, the membrane potential $u(t)$ drops to u_{rest}. In practice, when $t - t_0$ is much larger than τ_m, $u(t)$ can be roughly thought to have dropped to u_{rest}.

One useful conclusions that can be drawn from Equation 2.9 is that the absolute decay speed of the membrane potential slows down:

$$\frac{u(t-\Delta t) - u(t)}{\Delta t} > \frac{u(t) - u(t+\Delta t)}{\Delta t}, \ for \ t - \Delta t > t_0. \tag{2.11}$$

Constant current injection

Considering an LIF neuron with the initial membrane potential $u(t_0)$ and the input current $I(t)$ as shown below:

$$u(t_0) = u_{rest}, \tag{2.12}$$

and

$$I(t) = I_0. \ for \ t > t_0 \tag{2.13}$$

The solution of Equation 2.5 with these initial conditions is

$$u(t) - u_{rest} = RI_0[1 - e^{-\frac{t-t_0}{\tau_m}}], \ for \ t > t_0. \tag{2.14}$$

When $t = \infty$, Equation 2.14 becomes

$$u(t) - u_{rest} = RI_0. \tag{2.15}$$

This suggests that when time t is long enough, the membrane potential $u(t)$ will stabilize at a certain voltage which is determined by the input constant current I_0 and the resistance R in the LIF neuron.

A useful conclusion that can be drawn from Equation 2.14 is that the absolute charge speed of the membrane potential slows down:

$$\frac{u(t) - u(t-\Delta t)}{\Delta t} > \frac{u(t+\Delta t) - u(t)}{\Delta t}, \; for \; t - \Delta t > t_0. \tag{2.16}$$

In other words, as $u(t)$ gets closer to the maximum membrane potential $u_{rest} + RI_0$, the charge speed of $u(t)$ slows down ($\frac{\mathrm{d}u}{\mathrm{d}t}$ gets smaller).

LIF Response curve with a constant current injection

Equation 2.14 is the solution of the subthreshold equation of an LIF neuron when applying a constant current injection. This equation shows how the membrane potential $u(t)$ increases with time, and indicates that the highest value the membrane potential $u(t)$ can reach is $u_{rest} + RI_0$. Note that Equation 2.5 and its solution Equation 2.14 do not involve any spike generating mechanism. Instead, the spike generating mechanism is controlled by the supra-threshold equation 2.6. To explore how many time steps until a spike is generated and how many spikes will be generated in a given time window, Equation 2.14 and Equation 2.6 need to be considered together.

Assigning $u(t) = u_{threshold}$ and bringing it to Equation 2.14, the time t can be calculated. (Of course, if the highest membrane potential this LIF neuron can get is smaller than the threshold, which is $u_{rest} + RI_0 < u_{threshold}$, the membrane potential will never surpass the threshold, and Equation 2.14 will not have a solution. For these situations, t can be thought as infinite.) This time point is when the membrane potential reaches the threshold and causes a spike to be emitted. Also, an inter-spike interval (ISI) $t - t_0$ can be derived, which represents how long it will take for a spike to be generated with the constant current injection and the initial membrane potential u_{rest}. The firing rate (or firing frequency) of an LIF neuron is represented by $\frac{1}{t-t_0}$. Note that this calculation ignores the impact of the refractory time. If taking the refractory time into considerations. ISI is $t - t_0 + t_{refractorytime}$ and the firing rate is $\frac{1}{t-t_0+t_{refractorytime}}$.

Given a different constant input current RI_0, a corresponding firing rate can be calculated according to the method above (If the value t is infinite as aforementioned, the firing rate is approximated as 0 Hz). This input-output mapping is called the response curve of the LIF neuron. A typical response curve of an LIF neuron is shown in Figure 2.5, along with the input current and the change of membrane potential with time. Note that the time window of the input current injection in this Figure is about 1000 ms instead of infinite, so its output firing rate is discrete instead of continuous.

It is clear that the slope of this response curve gradually decreases with increasing input current. This response curve can be characterized by two values. The first value is the minimum input current that can give rise to an output spike in this spiking neuron, which is $\frac{u_{threshold}-u_{rest}}{R}$. This is when the maximum voltage of the membrane potential $u_{rest}+RI_0$ just equals the spiking threshold $u_{threshold}$. The second value is the cut-off frequency, which is the maximum output firing rate that can be reached by this LIF neuron. This value is primarily controlled by the refractory time.

LIF response curve with a noisy current injection

When the input is noisy, e.g. by adding Gaussian noise to the constant current input, the response curve of an LIF neuron will be smoothed as shown in Figure 2.6. Compared with constant input, noisy input can give rise to output spikes even with a small input current. This feature is considered to be an advantage of noise in SNNs, that is improving the sensitivity of weak signals and enabling these weak signals to be transmitted to other neurons by generating a few spikes [FSW08]. Nevertheless, a shortcoming is that these generated output signals are noisy rather than precise.

Generally, most input currents to SNNs are noisy, the reasons are twofold: First, the input current of an LIF neuron is the sum of input signals from many other neurons, these input signals are usually generated irregularly and they are discrete in time which causes the summed input current to be noisy. The spiking neurons in the input layer may receive constant input current, but these input neurons are only a small portion of the neurons in SNNs or in the nervous system. Second, the nervous system are affected by multiple noise sources [FSW08], which leads to noisy input currents to LIF neurons.

The response curve of an LIF neuron with noisy inputs has inspired the adoption of the ReLU activation function in artificial neural networks.

Instant current injection

The solution of a constant current injection is provided in Equation 2.14 which is shown below:

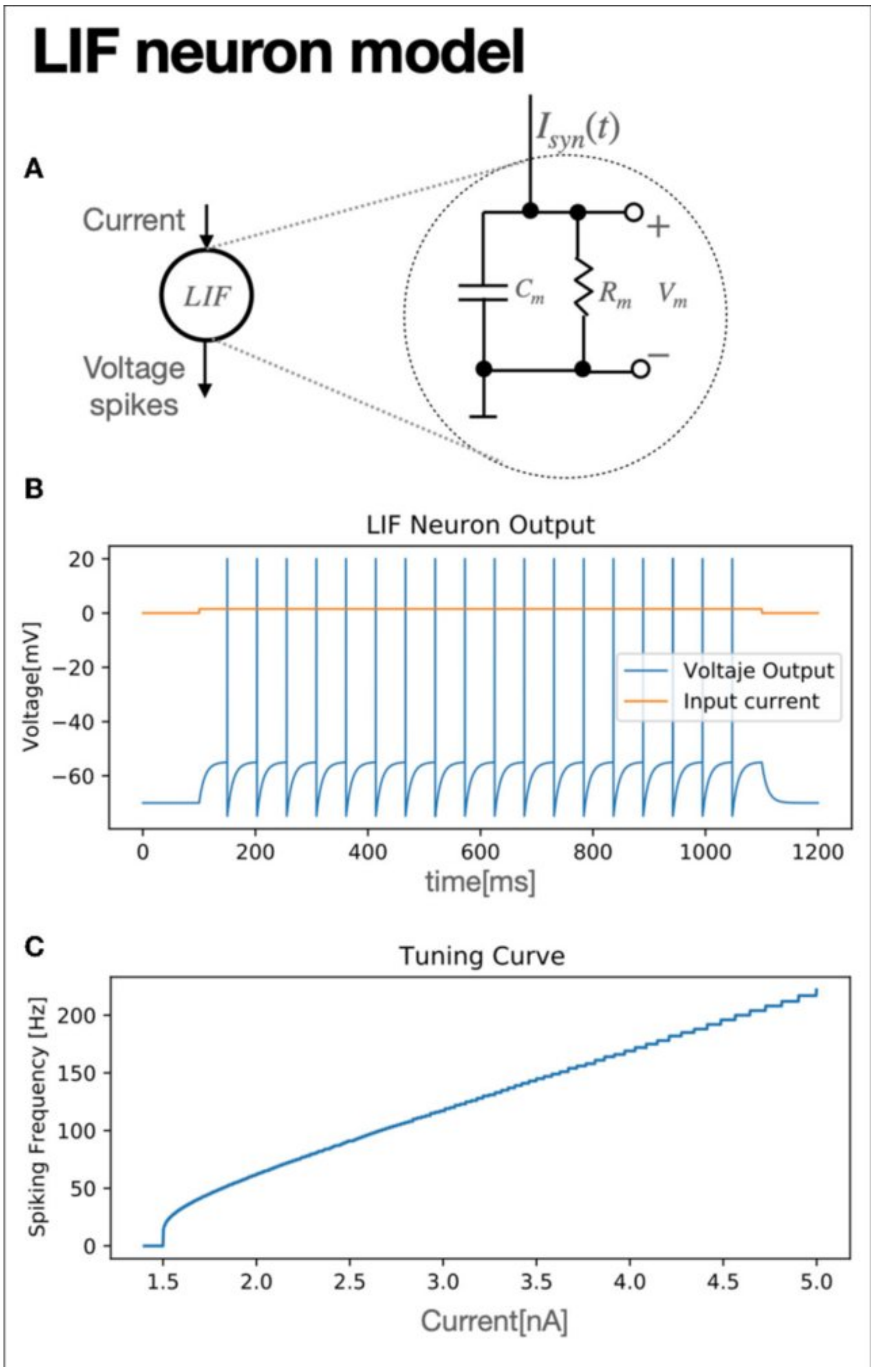

Figure 2.5: **A**. An LIF neuron and its RC circuit model. **B**. The constant current injection and how membrane potential changes with time. **C**. The response curve (tuning curve) of this LIF neuron. Figure is from [JLPPSRE22].

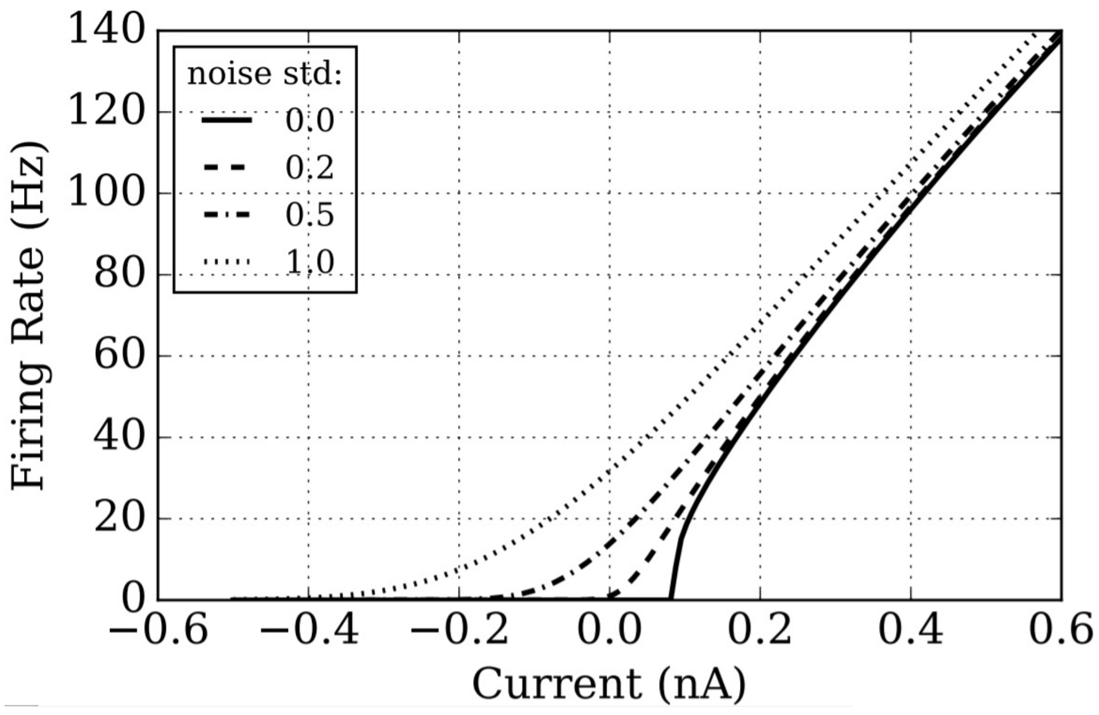

Figure 2.6: The response curve of an LIF neuron when inputs are noisy current injections. The noise type is Gaussian noise which is characterized by its standard deviation (whose values are 0, 0.2, 0.5, and 1.0 as shown in legend. 0 corresponds to a constant current injection). This shows that noisy inputs can smooth the response curve of an LIF neuron, and enable spikes to be generated when the input is weak and even negative. Note that the response curves of noisy inputs are noisy as well, so only some typical curves are picked here. Figure is from [LF16].

$$u(t) - u_{rest} = RI_0[1 - e^{-\frac{t-t_0}{\tau_m}}],\ for\ t > t_0. \tag{2.17}$$

When the input current I_0 is infinitely high and the current injection time $t - t_0$ is infinitely small, the input turns to instant current injection. This instant current injection can be defined by an equation similar to the Dirac delta function

$$I(t) = \begin{cases} +\infty, & for\ t_0 < t \leq t_0 + \frac{1}{\infty}, \\ 0, & for\ t > t_0 + \frac{1}{\infty}, \end{cases} \tag{2.18}$$

under the constraint of

$$\int_{t_0}^{+\infty} I(t)\mathrm{d}t = q. \tag{2.19}$$

Since the current injection time $t - t_0$ is infinite small, $e^{-\frac{t-t_0}{\tau_m}}$ in Equation 2.17 can be approximated by Taylor series:

$$e^{-\frac{t-t_0}{\tau_m}} = 1 - \frac{t-t_0}{\tau_m},\ for\ t = t_0 + \frac{1}{\infty}. \tag{2.20}$$

Note that only the first order of the Taylor series is considered here. Bring it to Equation 2.17, Equation 2.17 becomes

$$u(t) - u_{rest} = RI_0[\frac{t-t_0}{\tau_m}],\ for\ t = t_0 + \frac{1}{\infty}. \tag{2.21}$$

Since $I_0(t - t_0) = q$, and $\frac{R}{\tau_m} = \frac{1}{C}$. Equation 2.21 becomes

$$u(t) - u_{rest} = \frac{q}{C},\ for\ t = t_0 + \frac{1}{\infty}, \tag{2.22}$$

a very simple solution.

The advantage of modeling input current as an instant injection is that, compared with the constant injection, instant injection avoids computing membrane potential $u(t)$ for every time step. Also, the improved membrane potential given an input can be interpreted straightforwardly: it equals to $\frac{q}{C}$. Due to these reasons, some SNN simulations simplify the input as an instant injection. Also, when the time constant of synapses τ_{syn} (See Section 2.1.5 for the definition of τ_{syn}) is set much lower than the time resolution of simulation, the current transmitted by synapses can be seen as being injected to a neuron instantly.

Decomposing the input current

In Equation 2.5 which defines the subthreshold dynamics of the LIF model, the input current $I(t)$ is provided directly. This section gives more details on how $I(t)$ is calculated.

In SNNs, a spiking neuron usually receives inputs from many other spiking neurons, so the input current $I(t)$ is essentially the sum of currents sent from other spiking neurons. These spiking neurons are called presynaptic neurons, and their generated electric signals will go through synapses and are received by a postsynaptic neuron. These currents will cause postsynaptic potentials (PSPs) in the postsynaptic neuron, which are divided into the excitatory postsynaptic potentials (EPSPs) and the inhibitory postsynaptic potentials (PSPs), according to whether the postsynaptic potential increases (depolarization) or decreases (hyperpolarization).

The sum of these currents can be either spatial (when these currents are from different synapses), or temporal (when these currents are from the same synapse but arrive at different times), or both (when spatial integration and temporal integration happen simultaneously, known as spatiotemporal integration).

2.1.5 Synaptic model

Spiking neurons in the nervous system are connected and communicate through synapses. Nevertheless, not all neurons are connected, and how neurons in the brain are connected to realise highly-efficient information communication and information processing is a significant research topic. Different synapses have different strengths which are referred to as synaptic strengths, leading to different currents $I(t)$ aroused in postsynaptic neurons by identical spikes. The strength of synapses may be determined by the release probability of neurotransmitters, and the current jump caused by the release of neurotransmitters in synapses [Mur98]. Synapses are plastic, which is known as synaptic plasticity, and this is thought to involve the function of memory and learning in nervous systems. Two typical examples are STDP (spike-timing-dependent plasticity, a formulation of Hebbian learning) and structural plasticity.

In SNNs, a synapse is modeled by a synaptic weight ω. The sign of this weight represents the type of the synapse, one arouses EPSPs in the postsynaptic neuron and one arouses IPSPs in the postsynaptic neuron. The magnitude of this weight models the synaptic strength. Synaptic weight ω can contribute to the input current $I(t)$ of the postsynaptic neuron through two stages (exemplified by a single exponential synaptic model). First, a weight effect $\omega(t)$ in the single exponential synapse can be calculated by

$$\omega(t) = \sum_f \omega e^{-\frac{t-t_f}{\tau_{syn}}}. \tag{2.23}$$

t_f is the spike firing time of the presynaptic neuron whose axon is connected to this synapse, and these spikes can evoke the weight effect $\omega(t)$. τ_{syn} is the time constant of this synapse, and it controls the detailed synaptic dynamics, similar to the function of the membrane potential time constant τ_m in LIF neurons. Second, $\omega(t)$ can contribute to the input current $I(t)$ of the postsynaptic neuron. When applying a current-based synaptic model,

$$I(t) = \omega(t); \tag{2.24}$$

when applying a conductance-based synaptic model,

$$I(t) = g_{syn}(t)[u(t) - E_{syn}], \tag{2.25}$$

where E_{syn} is the reversal potential of a synapse (E_{syn} is usually set to -75 mV for inhibitory synapses and 0 mV for excitatory synapses) and $g_{syn}(t)$ is the conductance of the transmitter-activated ion channels whose value equals $\omega(t)$. Note that these equations only model one synapse and the temporal integration of $I(t)$. If there are more synapses connected to the postsynaptic neuron, the spatial integration of $I(t)$ needs to be considered as well.

When analyzing synapses at the neural-network level, more features of synapses can be found. Both theoretical analysis and experimental evidence indicate that excitatory synapses and inhibitory synapses are well balanced (E-I Balance) [ZY18]. E-I Balance is crucial for achieving effective computation and communication in the nervous system. Besides, it is claimed that there is a clear pattern on the synapses. Synapses connected from a given neuron are either inhibitory or excitatory but not blended; this is referred to as Dale's law or Dale's principle. From the computational perspective, it means that the synaptic weights of these synapses are either positive or negative. From the biological perspective, this synaptic pattern is because neurons release the same set of transmitters to all of their synapses. Some studies show that Dale's law is not a limitation for the computational capacity of SNNs [PAE08, TE16] and of ANNs[CKL+21].

2.2 Neuromorphic hardware

2.2.1 Categories of neuromorphic hardware

The concept of Neuromorphic hardware was proposed by Carver Mead, with the pursuit of electronic modeling of human neurology [Mea90]. Neuromorphic hardware replicates the merits presented in the nervous system on analog/digital circuits, with the goal of enhancing our understanding of the human brain and brain-inspired computing. It embodies the quote of the famous theoretical physicist Richard Feynman "What I cannot create, I do not understand". The brain merits that are replicated on neuromorphic hardware usually include parallel computing, asynchronous computing, event-based computing, near-memory computing, in-memory computing, domain-specific computing, dense connections, spike communications, and optimal spike routing. Other minor goals of neuromorphic computing can be categorized into two aspects: promoting academic research by providing efficient machines to computational neuroscientists and researchers in robotics for neuronal simulation, and promoting practical industrial applications by facilitating low-power low-latency deployments of spike-based deep learning algorithms and validating AI algorithms in new paradigms. Current neuromorphic hardware can be roughly divided into three types: analog, digital, and hybrid.

Analog neuromorphic hardware usually adopts energy-efficient devices such as memristors [LWM$^+$18], skyrmions [SJP$^+$20], and photonics [SBMN17], or utilizes subthreshold transistor dynamics to emulate neurons directly [Mea90], pursuing ultra-low-power computing. However, their building blocks are usually of low precision and suffer imperfection and noise, which imposes the challenges of reliable programming and large-scale applications.

Digital neuromorphic hardware simulates SNNs using digital circuits. "simulate" refers to the fact that in fully digital neuromorphic hardware, neuronal models, synaptic models, and spikes are all simulated digitally. Simulating these components may be less power-efficient than emulating them, but routing spikes with digital circuits is far simpler than with analog circuits. SpiNNaker [FGTP14], TrueNorth [MAAI$^+$14], and Loihi [DSL$^+$18] can be categorized as digital neuromorphic hardware.

Hybrid neuromorphic hardware mixes digital circuits and analog circuits to simulate SNNs. The spiking neurons and synapses in this type of neuromorphic hardware are emulated directly by analog components, and the generated spikes are routed to other neurons by digital circuits. Compared to pure analog neuromorphic hardware, the hybrid paradigm removes the obstacle of effective spike routing so it enables the building of larger machines. However, the imperfection problem still exists when emulating neurons and synapses. Neurogrid [KGT$^+$15] and BrainScales [SBG$^+$10] can be put into this category.

2.2.2 Other related hardware

Dynamic Vision Sensors (DVS). Dynamic Vision Sensors (DVS) [GDO$^+$20], also known as event cameras, are biologically inspired sensors that capture images asynchronously only when the pixel intensity changes. They are generally compared with static vision sensors applied in conventional frame-based cameras which capture images synchronously in a fixed interval. There are several differences between DVS and frame-based cameras: (1). DVS only measures motion (pixel intensity change) and acquires input signals in an event-based manner, rather than measuring absolute pixel intensity by frame-based signal acquisition. (2). DVS is achieved by more complicated circuits than static vision sensors, and each pixel in DVS usually occupies a bigger space than that in static vision sensors, so its spatial resolution is usually lower. (3). Though the spatial resolution is potentially lower, its temporal resolution is obviously higher. A typical latency of signal acquisition in DVS is 1 μs (10^{-6} s). The latency can be further reduced in lighter illumination. Hence, DVS gains more advantages in low-light environments than static vision sensors. The dynamic range of EVS is also wider. (4). DVS has lower bandwidth and lower power consumption. The acquired data is sparser and less than that acquired by static vision sensors. (5). Motion blur can be more efficiently alleviated.

When brightness changes in any pixel, DVS generates a piece of data that contains the information of the time, location, and the sign of brightness change. This spatial-temporal representation follows the address-event representation protocol, which is also known as AER [Mah92]. This data form is significantly different than the form of data gathered by conventional sensors, and it needs whole new algorithms to process. SNNs are naturally compatible with processing DVS data, and there have been many studies surging on this research topic [ONL$^+$13, FYC$^+$21b]. In the application aspect, DVS can work jointly with static vision sensors to capture more detailed information.

Dynamic audio sensors. They are biologically inspired sound sensors which measure air vibrations and generate neural signals [LvSMD13]. They mimic human hearing in the cochlea so they are also known as silicon cochleas. Similar to DVS, the data acquisition in dynamic audio sensors is asynchronous and event-based, and follows the AER protocol. In dynamic audio sensors, the "address" in the AER protocol is the frequency address instead of the spatial address used in DVS, e.g. the index of a channel which measures sound in a certain frequency range.

Neurorobotics. Robotics develops machines to substitute for humans and replicate human actions. Neurorobotics, or biologically-inspired robotics, is a research area in robotics, and it is closely related to many other research fields, such as computer vision, neuromorphic hardware, dynamic sensors, and SNNs. One promising application scenario is recording human behaviours by biologically-inspired dynamic sensors and processing the acquired data efficiently by SNN algorithms on neuromorphic hardware. The output of SNNs is then sent to a robot to generate motor commands for robot control. The actions of robot may in turn affect the behaviours of the human, which forms a closed-loop biologically-inspired system. Robotics can also be simulated in virtual environments.

2.3 Deep learning

This section introduces deep learning techniques related to SNNs, with the focus on neural networks and supervised learning.

Deep artificial neural networks, simplified as ANNs, are hierarchical neural network models that can learn input-output mappings from labeled examples. This learning process is called supervised learning as the examples are labeled, which is equivalent to providing supervision and instructions during learning. The learning in ANNs is essentially an information processing and extraction to represent infinite input-output mappings by finite network parameters trained by finite examples. “Finite” guarantees low cost during ANN training and “infinite” signifies the generalization of the ANN models.

2.3.1 Network architectures

The network architecture has a major impact on the performance that can be achieved after training. Some network architectures are intrinsically easier to train, or more computationally efficient, or more efficient to represent features during learning. This section introduces three typical network architectures, fully-connected networks, VGG, and ResNet.

Fully-connected networks usually contains several layers, and two adjacent layers are fully connected to each other. The connections between these adjacent layers are dense, which prevents this network architecture from scaling to more layers and solving more difficult tasks. VGG is a group of efficient image recognition models whose building blocks are convolutional layers and pooling layers. The information in these networks can only be propagated forward, and these networks do not contain any feedback connections. ResNet introduces shortcut connections over some layers, enabling establishing deeper ANNs and rendering better performance.

2.3.2 Objective functions

An objective function, also known as a cost function, or loss function, quantifies the performance of ANNs on a task, and provides a criterion for choosing an ANN learning process. During ANN training, the training samples X are sent to the input layer of an ANN which generates inference results Y' in the output layer. Each training example contains a label Y, and the goal of ANN learning is to minimise the difference between Y' and Y while ensuring this difference is small for new samples as well. There are various objective functions to choose from, such as MSE (mean squared error) and cross-entropy.

2.3.3 Learning rules

ANNs do not learn directly from humans or human knowledge, but learn by the pre-defined learning rules that control how their network parameters are updated. In supervised learning, the parameter updates are controlled by learning signals which eventually come from the objective function and are transmitted to these parameters by the backpropagation algorithm.

The backpropagation (backprop, BP) algorithm is widely used to facilitate the training of deep artificial neural networks. It gives a methodology to calculate gradients of the learning signal to the trainable parameters in a deep ANN. The calculation of gradients in the output layer is straightforward, and BP is applied to calculate the gradients in hidden layers and determines how gradient flows go back to the top layers.

2.3.4 Activation functions

Artificial neurons are the information processing units in ANNs, and their neuronal dynamics can be described by their activation functions. An activation function gives a mapping from an input x to an output y in a neuron which is usually nonlinear and differential. The nonlinearity of activation functions is vital for their computing capacity, and their differentiable form benefits the use of backpropagation to calculate gradients.

The rectified linear unit (ReLU) is one of the most frequently used activation functions, and it is inspired by the response curve of spiking neurons [GBB11]. ReLU can be approximated more easily by spiking neurons than other activation functions. The form of ReLU is shown as follows:

$$y_j = max(0, \sum_i \omega_{ij} y_i), \tag{2.26}$$

where y_j is the output of an artificial neuron, and y_i is the output of an artificial neuron in the previous layer. ω_{ij} denotes the synaptic weight from neuron i to j.

2.4 Deep spiking neural networks

2.4.1 Synaptic models

The synaptic models in deep SNNs are simpler than those in classical SNNs. Usually, the detailed synaptic dynamics are ignored, and the weight effect is fixed as

$$\omega(t) = \omega. \tag{2.27}$$

Note that $\omega(t)$ will be integrated into the postsynaptic neuron only when a spike is generated in the presynaptic neuron. These synaptic models in deep SNNs do not distinguish the type of synapses implicitly, but these synaptic models are essentially closer to the form of current-based synaptic model (Equation 2.24).

2.4.2 Neuronal models

There are a variety of neuronal models used in deep SNNs. In the context of ANN-to-SNN conversion, the focus of this thesis, there are three typical models as introduced below. In these neuronal models, the current will be injected to neurons instantly, which gives simplified equations of neuronal dynamics.

Leaky integrate-and-fire model. SNNs that use the leaky integrate-and-fire model usually feature high biological plausibility. LIF neuronal dynamics are described in Equation 2.28 and 2.29. The leak mechanism in this neuronal model hurts the similarity of ANN activation and SNN output response, which hinders this model from scaling to large datasets such as ImageNet.

$$\boldsymbol{u}_j^t = \alpha \boldsymbol{u}_j^{t-\Delta t} + \sum_i w_{ij} \boldsymbol{z}_i^t. \tag{2.28}$$

$$\boldsymbol{z}_j^t = 1 \; and \; \boldsymbol{u}_j^t = \boldsymbol{u}_{rest}, \quad when \; \Theta(\boldsymbol{u}_j^t - \boldsymbol{u}_{threshold}) = 1. \tag{2.29}$$

In these equations, $\boldsymbol{u}_j^t$ and $\boldsymbol{u}_j^{t-\Delta t}$ are the membrane potential of spiking neuron j at time t and $t - \Delta t$ respectively. α controls the leak rate, Δt is the time resolution during simulation. w_{ij} are the synaptic weights between neuron i in the previous layer and neuron j in this layer. $\boldsymbol{z}_i^t$ denotes the spike generated by neuron i at time t, and it has two states $\{0, 1\}$ representing whether or not a spike is elicited. The range of the sum is all spiking neurons connected to neuron j. $\boldsymbol{u}_{threshold}$ and $\boldsymbol{u}_{rest}$ are the threshold and the reset membrane potential of the spiking neuron j respectively. $\Theta(\cdot)$ denotes the Heaviside step function.

The standard integrate-and-fire model. The neuronal dynamics of this model are controlled by two equations below:

$$\boldsymbol{u}_j^t = \boldsymbol{u}_j^{t-\Delta t} + \sum_i w_{ij} \boldsymbol{z}_i^t, \tag{2.30}$$

$$\boldsymbol{z}_j^t = 1 \; and \; \boldsymbol{u}_j^t = \boldsymbol{u}_{rest}, \quad when \; \Theta(\boldsymbol{u}_j^t - \boldsymbol{u}_{threshold}) = 1. \tag{2.31}$$

Compared with LIF model, this neuronal model is more effective when applying to ANN-to-SNN conversion so more frequently adopted. Some studies have reported the competitive results on nontrivial datasets such as CIFAR-10 and ImageNet using this model.

The modified integrate-and-fire model (soft-reset integrate-and-fire model). To improve the compatibility to the ANN-to-SNN conversion technique, the reset mechanism of the standard integrate-and-fire model is changed from reset-to-rest (Equation 2.31) to reset-by-subtraction:

$$\begin{aligned} \boldsymbol{z}_j^t = 1 \; and \; \boldsymbol{u}_j^t = \boldsymbol{u}_j^t - (\boldsymbol{u}_{threshold} - \boldsymbol{u}_{rest}), \quad when \\ \Theta(\boldsymbol{u}_j^t - \boldsymbol{u}_{threshold}) = 1. \end{aligned} \tag{2.32}$$

This modification prevents the information loss brought by reset-to-rest, and enables more precise information to be be processed in spiking neurons and propagated to deeper layers. The subthreshold neuronal dynamics of this model is controlled by 2.30.

The SNNs with this neuronal model have achieved the highest accuracy in various datasets.

2.4.3 Training

Deep SNNs can be trained in various ways, and there are many new training approaches proposed each year. This section starts from introducing two basic training methods: ANN-to-SNN conversion and direct training. They are both based on ANN training techniques, while other SNN training methods are primarily based on these two methods. Some training techniques will apply feedback alignment to avoid backpropagation, which is introduced as well.

Training approaches

- **ANN-to-SNN conversion**. When applying rate coding in SNNs, the response curve of a spiking neuron is similar to a ReLU activation function in ANNs. Hence, an SNN can be implicitly trained through training an ANN whose activation function is ReLU and converting this ANN into a rate-coded SNN, an approach referred to as ANN-to-SNN conversion [DNB$^+$15]. The fundamental problem in ANN-to-SNN conversion is how to effectively map ANN activation to SNN firing rate to obtain higher accuracy, lower latency, and lower computational burden.

- **Direct training by surrogate gradients**. Direct training is inspired by the backpropagation through time algorithms in recurrent neural networks. When conducting direct training, an SNN is unfold in the time domain, and the learning signal is backpropagated spatiotemporally [NMZ19]. Direct training usually renders lower latency than ANN-to-SNN conversion. Also, direct training is naturally compatible with DVS data.

- **Feedback alignment**. Feedback alignment [LCTA16] is an alternative to backpropagation. Specifically, the feedback weight matrices are randomly generated, and the learning signal will transmit through these weight matrices without the need to access the original weight matrices in ANNs. This algorithm avoids the weight mirroring problem in backpropagation and is claimed to be more biologically plausible. On the other hand, the randomly generated weights are biased compared to the original weights matrices, causing a biased learning process and lower performance especially in challenging datasets such as ImageNet. Note that feedback alignment is not a learning approach, e.g., it does not generate learning signals by itself. It needs to associate with other learning methods such as gradient descent to realize its function.

- **Online learning**. Online learning attracted wide scholarly attention, as it promises to complete learning on new input samples while processing them. In other words, the training and inference happen simultaneously. In contrast, offline learning cannot achieve such real-time training but relies on extra time to train on new samples. These samples will accumulate and require external memory to store them, waiting to be loaded and trained again when the machine is spare. These features of online learning—simultaneous learning and low memory requirements—facilitate on-chip learning, especially learning on neuromorphic chips [IC20]. Note that most online learning research in SNNs is not continuous learning, which suggests that they do not solve the problem of catastrophic forgetting. An online learning algorithm called e-prop is exemplified here [BSS$^+$20]. In e-prop, the equation of backpropagation through time is divided into two parts: the eligibility trace which originates from a reinforcement learning concept, and the instantaneous learning signal which ignores the learning signal from future time steps. Both of these two parts can be obtained during the inference of input data without the need to wait for future signals, promoting e-prop as a powerful algorithm on recurrent SNNs. e-prop can be combined with feedback alignment to avoid accessing weight metrics.

- **Event-based on-chip direct training**. It is spiking neurons rather than artificial neurons that are simulated or emulated on neuromorphic hardware. Hence, the on-chip learning algorithms on neuromorphic hardware are training spiking neurons. Considering this, the direct training algorithms, instead of ANN-to-SNN conversion, are more compatible with on-chip learning. One primary requirement of on-chip learning is that the information involved in learning should be local, since accessing global information is expensive [TKPM21]. It suggests that on-chip learning algorithms can be combined with feedback alignment. Besides, on-chip learning needs to be power-efficient and compatible with neuromorphic hardware, or on-cloud learning would be more practical. How to make the learning signals in on-chip learning sparse and event-based for higher compatibility with neuromorphic hardware has triggered a surge of interest.

2.4.4 ANN-to-SNN conversion

This section provides more technical details and analysis on ANN-to-SNN conversion, as it has significant impact on deep SNN optimizations and it is at the center of many studies presented in this thesis.

ANN-to-SNN conversion is a highly effective training method. It is essentially training an ANN to represent a rate-coded SNN at the abstract level where only the input-output response curve of a spiking neuron is considered. In theory, if a perfect ANN-to-SNN mapping is achieved, an SNN built by ANN-to-SNN conversion can achieve exactly the same accuracy as that of the ANN. Some recent studies suggest that by adding some constraints during ANN training before ANN-to-SNN conversion, the SNN can obtain some extra features, such as sparser communications (lower firing rates) [SLBS20, NBLD22]. ANN-to-SNN conversion can transfer the progress in ANNs to promote building more competitive SNNs. Considering that the current ANN community is obviously larger and its research is prosperous, ANN-to-SNN conversion is expected to promote faster progress in SNNs, at least at this stage.

The ANN-to-SNN conversion technique is proposed in [CCK15] which shows using spiking neurons to approximate hyperbolic tanh neurons in ANNs. This study not only demonstrates good approximation between a single spiking neuron and an artificial neuron but also shows an unchanged approximation of these neurons at the network level, by which a functional SNN can be built by converting an already-built ANN. [DNB+15] propose to use ReLU in ANNs and show competitive accuracy on MNIST, probably benefiting from the better approximation of the response curve of spiking neurons to ReLU. The implicit link between the activation function in spiking neurons and artificial neurons is suggested in more early studies, see [OB11, GBB11] which introduce this in the perspective of neuroscience and deep learning respectively.

Central to ANN-to-SNN conversion are the adopted normalization strategies to map the ANN activation to the SNN firing rate. This topic is first researched in [DNB+15], and there are two normalization methods proposed in this paper, named model-based normalization and data-based normalization. Model-based normalization is too conservative which leads to a long latency; data-based normalization takes both the model parameters (such as synaptic weights) and the input data into consideration so renders a better latency and accuracy. The maximum firing rate of an SNN when applying data-based normalization is 1000 Hz under the time resolution of 1 ms. [RLH+17] provides the exact equations for data-based normalization as well as equations to normalize biases and convert batch normalization. The maximum firing rate defined in this paper is 1000 Hz as well, but it maps the great majority of ANN activation to higher SNN firing rates. As a result, the latency is reduced considerably. Besides, [RLH$^+$17] applies analog input (instead of rate coding) and the modified IF model (instead of the standard IF model) to render better ANN-to-SNN mapping, enabling to convert ANNs for challenging datasets such as CIFAR-10 and ImageNet. A channel-wise data-based normalization is raised in [KPNY20] to improve the granularity of parameter normalization. Consequently, the firing rate is higher than the original data-based normalization.

[SYW$^+$19] proposed a firing rate management strategy called Spike-Norm. This strategy is non-deterministic so the firing rate range is different for each trial. The firing rate of spiking neurons after applying this strategy is low in general, though the inference latency is extended consequently. The bias in ANNs is not modeled in SNNs in this research. [LF21] proposed an approach to enable more flexible firing rate management compared with data-based normalization. It shows that for the standard IF model, applying a maximum firing rate of 400Hz outperforms the default 1000 Hz used in [DNB$^+$15]. Also, the bias and batch normalization is successfully modeled in this study. [HSR20] illustrates the response curve of the standard IF model and suggests that this curve is non-linear. However, this non-linear shape of the response curve will only appear in the first hidden layer. Feedforward propagation will smooth this response curve in deeper layers [LF21].

The equations for data-based normalization are provided below:

$$\widetilde{w_n} = w_n * \frac{\lambda_{n-1}}{\lambda_n}, \tag{2.33}$$

$$\widetilde{b_n} = b_n * \frac{1}{\lambda_n}. \tag{2.34}$$

w_n and b_n ($\widetilde{w_n}$ and $\widetilde{b_n}$) are the weights and biases before (after) the normalization in layer n respectively. λ_n is the maximum ANN activation value in layer n.

2.4.5 Encoding

Rate coding. Rate coding was initially found in muscle nerve cells [AZ26], and was further expanded to brain neurons as more extensive biological evidence appeared. The information in rate coding is encoded by the firing rate—the generated spike counts in a certain time window—to guarantee the ratio of spike count $N(spikes)$ and time window

Δt is proportional to input density $Density(Input)$ as below:

$$\frac{N(spikes)}{\Delta t} \propto Density(Input). \tag{2.35}$$

The time window Δt usually needs to be long enough to maintain sufficient encoding precision. The long time window is not suitable for tasks that require a fast response. However, it provides enhanced robustness to noise, e.g. noise caused by accidental appearance or loss of spikes. Rate coding was applied in SNNs to solve various pattern recognition tasks[BFD+21, LF21].

The naive way to encode information according to Equation 2.35 is using pure rate coding. In pure rate coding, the inter-spike interval (ISI) $\frac{\Delta t}{N(spikes)}$ is fixed. However, pure rate coding cannot be effectively applied in SNN simulators and on neuromorphic hardware. This is because pure rate coding requires a high time resolution to calculate the spike emitting time according to ISI, but the time resolution in these platforms is usually low (Typical time resolutions are 1 ms or 0.1ms). This mismatch will bring considerable errors in neural encoding.

Most rate coding applied to SNN neuromorphic hardware, SNN simulators and SNN algorithms is the stochastic version of the rate coding. Stochastic rate coding introduces randomness and probability to neural encoding, based on the fact that spike patterns are different in each neuroscience experimental trial, even when the input stimuli is kept the same. Due to this randomness, the generated spike counts in a certain time window in statistical rate coding is not guaranteed to be equal to the target spike count $N(spikes)$. Nevertheless, its error is still smaller than that in pure rate coding. In statistical rate coding, whether a spike is generated in a simulation time step is controlled by the equation below:

$$P(S_t = 1) \propto Density(Input). \tag{2.36}$$

$S_t = 1$ represents a spike generated at time t. $P(S_t = 1)$ is the probability of generating a spike at time t, which is proportional to the input intensity $Density(Input)$. According to Equation 2.36, whether a spike is generated in one time step follows a Bernoulli distribution. Also, whether a spike is generated in each time step is independent, so the overall generated spike counts in n time steps follows a Binomial distribution.

In addition to the lower error in neural coding, another useful advantage of stochastic rate coding over pure rate coding is that it does not require defining the time window Δt in advance. This is due to stochastic rate coding calculating spike generation in each individual time step instead of based on a predefined time window.

Temporal coding. Temporal coding encodes information in the precise time of a spike or the time interval between spikes. It is a highly efficient encoding approach that is capable of providing richer information than rate coding. However, naive temporal coding is not robust to noise such as the variation of spike generation time. There are many temporal coding schemes proposed in SNN research, such as rank order coding [GJDSA07], latency coding [PWZ$^+$19] and Time-to-First-Spike Coding [PKNY20].

Population coding. Population coding [PWZ$^+$19] encodes information in the firing pattern of a group of neurons. In population coding, each neuron responds to a part of the input signal, so the joint response activity (spikes) of a neuronal population can be combined to encode the original input signal.

Binary coding. Binary coding [SM19, SM21] encodes information according to binary code, a coding scheme used in computer processing. In binary coding, the generation of a spike corresponds to the signal "1" and the absence of a spike corresponds to the signal "0". Each spiking neuron represents a bit in binary code.

Burst coding. In burst coding [GFES21], a burst of spikes is sent out instead of one spike. Sending more spikes can improve the robustness to noise and non-idealities, making burst coding a useful encoding scheme on analog or hybrid neuromorphic hardware.

2.4.6 Datasets

Datasets are a crucial factor behind the prosperity of deep learning, while it is also significant to SNNs to test and compare the effectiveness of SNN algorithms and drive more rapid progress. The datasets adopted in SNNs can be roughly divided into two categories: datasets inherited from deep learning such as some pattern recognition datasets, and datasets built for SNNs or datasets intrinsically suitable for using SNNs to solve such as DVS datasets.

Deep learning datasets

- **MNIST**. MNIST is a handwritten digit database for computer vision and pattern recognition [LBBH98]. It has a training set of 60,000 examples and a test set of 10,000 examples. These examples are 28x28 pixel pictures of handwritten numbers between 0 to 9. Each pixel has a grey-scale value between 0 (black background) and 255 (white foreground).

- **Caltech101**. Caltech101 [FFFP04] contains about 9,000 object images grouped in 101 classes, plus a background class. The size of these images varies, with a typical height of 245 pixels and a width of 302 pixels.
- **fashion-MNIST**. This dataset [XRV17] shares the same sample size, data format, and dataset splits of training samples and test samples as MNIST: It consists of 60,000 examples in the training set and 10,000 examples in the test set. Each example is a 28x28 grey-scale image. There are 10 classes of clothes in Fashion-MNIST, such as “shirt” and “sneaker”.
- **EMNIST**. EMNIST [CATVS17], or Extended-MNIST, is a handwritten character and digit dataset whose image format and dataset structure are arranged in a similar way to MNIST. There are six different subset splits in this dataset such as the ByClass dataset and the Balanced dataset.
- **SVHN**. SVHN [NWC$^+$11] is the abbreviation of Street View House Numbers, a real-world image dataset tailored for object recognition algorithms. Similar to MNIST, each sample in this dataset is a single digit whose label is one of “0” to “9”. Nevertheless, the size of each sample is 32-by-32 and the size of this dataset is bigger (73,257 digits for training, 26,032 digits for testing, and 531,131 additional less difficult samples which can be used as extended data in the training set.); also, the digit recognition problem in this dataset is more challenging than MNIST, since the digit is in a natural scene. SVHN is collected from house numbers in Google Street View images.
- **CIFAR-10**. This dataset [KH$^+$09] contains 60,000 images, divided into 10 classes such as bird and cat. There are 50,000 training images and 10,000 test images. The size of each image is 32x32 pixels and their format is RGB.
- **CIFAR-100**. This dataset [KH$^+$09] is similar to CIFAR-10 in image size and format (32x32 color images), but differs in the number of classes and the number of images per class (100 classes with 600 images each, divided into 500 training images and 100 test images.).
- **ImageNet**. ImageNet [DDS$^+$09] is a large object recognition dataset, and the frequently-used version is ILSVRC-2012. ILSVRC-2012 has over one million labeled RGB examples, and 1000 object classes. Generally, its validation set instead of the test set is adopted to test the model after training. The size of the validation set is 50,000.

- **Keyword spotting tasks**. Keyword spotting involves monitoring a real-time audio stream with the purpose of identifying keywords of interest in utterances. One typical application scenario is the wake-up word detection such as “Hey, Siri” in virtual assistants. A keyword spotting dataset provided in [BCHE19] consists of a training set of about 2,000 utterances and a test set of 192 utterances, collected from 96 speakers,

- **TIMIT**. TIMIT [ZSG90], or TIMIT Acoustic-Phonetic Continuous Speech Corpus, is a speech recognition dataset. The samples in this dataset are recorded from 8 dialects of American English, and each of them reads 10 phonetically-rich sentences

- **PASCAL VOC**. PASCAL VOC (PASCAL Visual Object Classes) [EEVG$^+$15] is a dataset for object recognition, semantic segmentation, and classification tasks, containing 20 categories such as vehicles and animals. Each sample in this dataset is an image with pixel-level segmentation annotations, bounding box annotations, and object class annotations. It has 1,464 images in the training set and 1,449 images in the validation set

- **TIDIGITS**. TIDIGITS [LD93] is an acoustic dataset originally collected at Texas Instruments, Inc, with the purpose of designing and evaluating algorithms for speaker-independent, sequence-connected audio classification. The audio signals in this dataset are labeled from ‘zero’ to ‘nine’ and ‘oh’, and there are 2,464 training and 2,486 testing utterances. Speakers are selected for balance (111 men, 114 women, 50 boys, and 51 girls), and each speaker group is divided into a training set or a test set.

- **RWCP**. RWCP (Real World Computing Partnership) [NHA$^+$00] is a sound dataset consisting of 200 training and 200 test samples. Unlike speech datasets such as TIDIGITS, this dataset includes non-speech sounds with the labels such as “bell” and “phone”.

Event-based datasets

- **POKER-DVS and SLOW-POKER-DVS**. The POKER-DVS [SGLB15] database comprises 131 poker pip symbols which are tracked and extracted from a dynamic vision sensor. The samples in this dataset are 32x32 pixel event streams, categorized into 4 categories (“club”, “diamond”, “heart” or “spade”) according to the poker symbol. The SLOW-POKER-DVS dataset is collected in the similar manner, while the poker cards are slowly moving across the screen.

- **MNIST-DVS and MNIST-FLASH-DVS**. The MNIST-DVS [SGLB15] database consists of 30,000 DVS recordings of handwritten digits between 0 to 9. The samples are collected by moving the digit samples in MNIST and recording them by a highly-sensitive DVS. MNIST-FLASH-DVS is collected in the similarly way to MNIST-DVS. However, Each digit is flashed several times during DVS recording, which make this dataset more challenging.

- **N-Caltech101**. This is a neuromorphic vision dataset converted from an existing Computer Vision static image dataset Caltech101 [OJCT15]. This dataset is captured by mounting a Asynchronous Time-based Image Sensor (ATIS) sensor on a pan-tilt unit, and move the sensor while viewing Caltech101 samples on an LCD monitor.

- **N-MNIST**. The Neuromorphic-MNIST (N-MNIST) dataset is a spiking version of the original frame-based MNIST dataset, captured in a similar way to N-Caltech101 [OJCT15]. It has the same dataset size and sample size as the original MNIST dataset: 60,000 training samples and 10,000 test samples, and 28x28 pixels per sample.

- **N-TIDIGITS**. This dataset is converted from the TIDIGITS dataset, an audio dataset, to spiking version using a spiking silicon cochlear sensor[ANDL18].

- **Yin-Yang**. The Yin-Yang dataset [KGP22] is tailored for deep SNN research especially that adopts biologically-plausible error backpropagation algorithms. The training set has 5,000 samples and the test set has 1,000 samples. Each sample is a two-dimensional representation of the yin-yang symbol, and it is classified into one of the three classes, “Yin”, “Yang”, and “Dot”.

- **DVS128 gesture dataset**. This is a spiking gesture recognition dataset, recorded by a DVS in the form of event streams [ATB$^+$17]. This dataset contains 11 hand gestures, collected from 29 subjects under 3 illumination conditions

- **SHD**. SHD [CSSZ20], the abbreviation of Spiking Heidelberg Digits, is an audio classification dataset. It is converted from the Heidelberg Digits dataset to the spiking version. There are 8,332 training samples and 2,088 test samples, each sample has 700 input channels. These samples are recordings of spoken digits from 0 to 9 from 12 speakers in both German and English.

- **SSC**. SSC [CSSZ20]. is converted from the Speech Commands dataset [War18], which was initially released by Google, to the spiking version. It consists of utterances recorded from a great number of speakers under less controlled conditions, divided to 35 categories.

- **N-Cars**. The N-CARS dataset [SBB+18] is a large-scale real-world event-based dataset for car classification. It was directly recorded in urban environments by an event-based sensor. The raw data is an approximately 80-minute video, which is then extracted and processed to generate samples. These samples are grey-scale images, including 12,336 car samples (7,940 samples in the training set and 4,396 in the test set) and 11,693 background samples (7,482 samples in the training set and 4,211 in the test set). Each sample lasts 100 milliseconds.

- **CIFAR10-DVS**. This dataset [LLJ+17] is converted from CIFAR-10, a frame-based object classification dataset. A subset of samples in CIFAR-10 is converted to event streams with the size of 128×128 by a dynamic vision sensor, and there are totally 10,000 samples in this dataset.

- **ES-ImageNet**. ES-ImageNet [LDQ+21] is an event-stream (ES) version of a computer vision dataset ImageNet, comprising about 1,300,000 event-based samples divided into 1,000 categories. This dataset is generated by an algorithm (which is called Omnidirectional Discrete Gradient) rather than acquired by a DVS, which enables building such a large-scale dataset with low-cost and high-speed.

2.4.7 SNN simulators

SNNs are eventually targeted to be deployed to neuromorphic hardware to achieve brain-like energy efficiency for practical tasks. Before that, SNNs can be simulated on simulators for concept validation and fast development. The SNN simulators introduced in this section are crudely divided into two main categories: classical SNN simulators which are initially designed to run classical SNNs, and the simulators built upon deep learning tools for deep SNN simulations.

Classical SNN simulators

- **NEST**. NEST [GD07] is a function simulator for spiking neural network models. This simulator emphasizes the function and dynamics at the neuronal level and the connection and structure at the neural system level. The exact morphology of individual neurons is ignored. NEST supports the simulation of some learning and plasticity mechanisms in SNNs.

- **Brian and Brian2GeNN**. Brian [GB08] is an efficient Python-based SNN simulator with the goal of providing easy-to-use SNN modeling tools for scientists in computational neuroscience and other research disciplines. It is flexible and scalable, which enables new models, especially networks of single-compartment spiking neurons (point model), to be developed rapidly. Brian has realized the second version called Brian2 [SBG19]. Brian2GeNN [SGN20] couples Brian and GeNN to accelerate SNN modeling by using GPUs. One to two orders of magnitude times speed-up is realized in two example models on Brian2GeNN.

- **PyNN**. PyNN [DBE$^+$09] is a python package for SNN simulation. It is simulator-independent, which means a model defined by PyNN APIs can be run on any supported simulators by PyNN such as NEURON, NEST, and Brian, and on neuromorphic hardware including SpiNNaker and BrainScaleS. PyNN can be seen as a high-level SNN model library to define SNN elements including neurons, layers, connections, and so on at the abstract level. Some low-level APIs are also provided in PyNN, giving more flexibility and, potentially more efficiency.

Simulators based on deep learning tools

- **PyTorch**. PyTorch [PGM$^+$19] is an open-source machine learning framework, aiming to accelerating model prototyping as well as practical hardware deployment. It can build ANNs with very deep network structures efficiently and the gradients, a key factor related to the learning process in ANNs, can be calculated automatically. PyTorch supports GPU acceleration which improves the simulation speed of deep neural networks by orders of magnitude compared with CPUs-based simulations. Some studies simulate SNNs in PyTorch when the investigated tasks, learning algorithms, and network structure are closely related to deep learning [WDL$^+$18, LF21]. PyTorch provides many advantages for SNN simulation. It is convenient to build an SNN with typical ANN network topology. Also, the learning algorithms of SNNs are well supported in PyTorch: For SNNs built by ANN-to-SNN conversion, the learning process takes place in ANNs. This ANN learning process can easily be achieved in PyTorch since PyTorch is designed for ANNs and ANN learning; for SNNs built by direct training, the training is based on the surrogate gradient algorithms which needs to modify gradients. This need on custom gradient modifications can be met by PyTorch as well. On the other hand, simulating SNNs on PyTorch suffers some difficulties. For example, PyTorch does not provide standard spiking neuronal models to use, so any spiking neuronal models need to be built from scratch. Some functions in spiking neuronal models such as leak and refractory time may not be straightforward to simulate. Also, some synaptic connection patterns such as random connection and some synaptic dynamics such as postsynaptic potential which are naturally supported in classical SNN simulators may not be supported in PyTorch by default. Another difficulty is that the recording of neuronal states, such as spike count recording and membrane potential recording in each time step, need to be programmed manually from the beginning, which involves many data merge, data format conversion, and data calculation issues.

- **SpykeTorch**. This [MGNDM19] is a an open-source high-speed PyTorch-based simulator for convolutional SNNs. Each spiking neuron in this framework only generates one spike at most and input is encoded by rank order to build highly-efficient SNNs. This framework supports the STDP learning rule and reward-modulated STDP learning, and can run on GPUs for acceleration.

- **CARLsim**. CARLsim [BCC$^+$15] is a library for simulating large-scale SNNs efficiently while keeping high biological details. For example, this simulator allows building SNNs composd of Izhikevich spiking neurons and realistic synaptic dynamics and accelerating them using GPUs. It provides C/C++ level interface like in PyNN, which improves the flexibility of SNN modeling in this simulator.

- **SpikingJelly**. SpikingJelly [FCD$^+$20] is an open-source framework for deep SNN simulation on PyTorch. Some high-impact SNN algorithms and some neuromorphic datasets are embedded in this framework, to move obstacles on PyTorch-based SNN simulation.

- **snnTorch**. This [EWN$^+$21] is a PyTorch-based Python package with the function of conducting gradient-based learning in spiking neural networks. snnTorch predesigns many spiking neuronal models such as leaky integrate-and-fire neurons as well as other functions including rate coding, the surrogate gradient descent algorithm, and the online learning algorithm, which empower user-friendly deep SNN modeling.

- **Rockpool**. Rockpool is a Python package to build machine-learning-based SNNs for signal processing applications. It contains necessary functions to build, simulate, train, test, and deploy SNNs. Rockpool is simulator-independent so the SNNs built in Rockpool can be run on simulation backends such as Brian2, NEST, and Torch. It also supports SNN deployment on event-driven neuromorphic hardware.

- **DECOLLE**. DECOLLE [KMN20] (a rough abbreviation of Deep Continuous Local Learning) is an SNN simulation framework for online learning. This framework provides a local error signal to enable online learning in SNNs with no extra memory overhead, paving the way to further deployment on neuromorphic hardware

- **Norse**. Norse [PP21] is a deep learning library for spiking neural network simulation with the aim of exploiting the advantages of bio-inspired neural components such as sparsity and event-based computing. Also, since Norse is based on PyTorch, the merits of deep learning tools can be retained as well. It provides some examples to solve MNIST and CIFAR-10 using SNNs as well as to realize some learning algorithms in SNNs.

- **BindsNET**. BindsNET [HSK+18] is a Python package for SNN simulation, utilizing PyTorch Tensor-based computation on CPUs or GPUs. It assists research into applying SNNs to machine learning problems.

- **cuSNN**. cuSNN [PVSDC20] is a C++ library to simulate large-scale SNNs which are accelerated by GPUs. It contains some LIF models and STDP learning rules, and supports building SNNs with convolution network topology.

- **Nengo**. Nengo [BBH+14] is a Python package for building, evaluating, and deploying neural networks including their spiking versions. Nengo is based on the Neural Engineering Framework, it is highly flexible and easily extensible. The detailed neuronal models, learning rules, and so on can be customized by users.

- **Sinabs**. Sinabs is a python library for developing and implementing of Spiking Convolutional Neural Networks (SCNNs), managed by SynSense (former aiCTX AG).

- **SLAYER**. SLAYER [SO18] is the crude abbreviation of Spike LAYer Error Reassignment, and it provides methods to handle the non-differentiability of spike function and to achieve effective SNN training. Both the fully connected network and convolutional neural network are available to be trained in this framework.

- **SNN toolbox**. SNN toolbox (Spiking neural network conversion toolbox) is designed for automating ANN-to-SNN conversion. An ANN built and trained in PyTorch or in other deep learning frameworks can be converted to SNNs by this toolbox, and run on SNN simulators or neuromoprhic hardware.

2.5 Summary

This chapter introduced the fundamental knowledge about deep SNNs as well as related concepts in classical SNNs, neuromorphic computing, and deep learning. The computational property of a single spiking neuron was highlighted in Section 2.1. Also, the detailed equations that describes the neuronal dynamics and synaptic dynamics were provided. Neuromorphic hardware was initially designed to conduct bio-inspired computing and enhance our understanding of the human brain. However, more deep learning elements are now involved in neuromorphic computing. Deep SNNs are promising to achieve power-efficient event-based computing. Section 2.4 described the building blocks of deep SNNs.

Chapter 3

SNNs on neuromorphic hardware

3.1 Introduction

So far, SNNs are still in the early stage, and many researchers are attempting to discover the advantages of SNNs compared with ANNs which are currently more successful on many benchmarks [LBBH98, KH$^+$09, RDS$^+$15]. Two potential advantages of SNNs have been found and received considerable scholarly attention: First, SNNs can achieve better energy efficiency when power-efficient neuromorphic hardware is applied [MPSC17], and when SNNs are optimized for short latency [WCZ$^+$19] and low firing rates [PSRGSGLB20]. Second, a rate-coded SNN can generate outputs faster than an equivalent ANN, though its output may be noisy and of low precision. The inference precision would then rise with more evidence accumulated over time by the spiking neurons [DNB$^+$15]. This feature is potentially useful for dealing with the challenge of real-time processing in self-driving vehicles which current ANNs struggle to overcome. In summary, SNNs have potentially beneficial effects on power efficiency and fast inference. However, we have just scratched the surface of realizing these advantages [PP18] and we do not know yet how to use these characteristics properly to make SNNs more competitive than their ANN counterparts.

To promote a deeper understanding of SNN merits, this chapter first investigates one potential advantage on noise robustness of SNNs and compares it to ANNs. The results show that SNNs are more robust to Gaussian noise in synaptic weights, a typical perturbation to SNNs when deploying SNNs on cutting-edge materials, than ANNs under some conditions. This finding will expand our understanding of the neural dynamics in SNNs and the advantages of SNNs compared with ANNs. Also, the reported results imply the possibility of using high-performance cutting-edge materials with intrinsic noise as an information storage medium in SNNs, such as memristors and skyrmions. The contents related to the simulation of deploying the weights of SNNs on skyrmions are illustrated in the following section. At the end of this chapter, the process of deploying an SNN on SpiNNaker, a digital neuromorphic hardware, is presented.

3.2 Preliminary

3.2.1 Weights

At the algorithm level, the function of a deep neural network model is characterized by its network parameters, including weights, biases, and so on. The most fundamental parameters of a neural network are its weights which define the synaptic connections between two adjacent layers. The biases, on the other hand, are not a compulsory element in neural network models, such as in [DNB^{+}15]. The lack of biases in a neural network model may cause an accuracy degradation as the model representation capacity is impaired.

At the hardware level, weights attract more attention than biases as well. There is much research focusing on how to deploy a neural network more efficiently to hardware during inference. One promising approach is using the structure of a crossbar to store a weight matrix efficiently in an analog manner, a method called in-memory neural network computing.

In summary, the weights are widely researched both at the algorithm level and at the hardware level. The research presented in this chapter also focuses on noise injections into weights rather than biases.

3.2.2 Weights in hardware

The research into the storage medium of weights in deep neural networks is vigorous. Three typical storage mediums for weights are listed and discussed here: digital platforms, analog platforms, and non-volatile analog platforms.

Digital platforms are the main solution for parameter storage in neural networks. The neural network models are stored in digital memories and are loaded to digital computing platforms such as GPUs, CPUs, TPUs, and FPGAs during inference. GPUs and other similar computing devices are reliable enough and the neural network models can be run on them without the need to consider noise. Keeping these models on digital platforms, however, will consume significant electricity, which poses a challenge to power-efficient neural network inference for real-world applications. This drawback is more serious when the input to the neural network is sparse.

Analog platforms are more power-efficient for neural network applications. There are different ways to represent weights in analog platforms, such as using an analog crossbar and by digital SRAM. Some variations may exist in these weights for both cases. When conducting feedforward propagation, these weights are extracted from these storage media and participate neural network computing in the form of current.

Current neural network storage techniques do not result in noisy weights, but noise could become an issue when more advanced storage devices such as memristors and magnetic skyrmions are used to implement neural networks [SSSW08, JUZ$^+$15]. These two devices have the excellent characteristics of non-volatility and nanoscale size. These advantages make memristors and magnetic skyrmions good candidates to deal with the challenges of high power-dissipation in neural networks and the continuation of Moore's law. Memristors and magnetic skyrmions have been applied to SNNs by both theoretical simulations and experimental investigations [SHY18, CSR18]. However, these cutting-edge devices may contain non-negligible random noise at room temperature, which is typically Gaussian distributed.

3.2.3 Noisy weights

The noise type investigated in this study is Gaussian noise. The key parameters of Gaussian noise are its mean and standard deviation (SD). The mean of Gaussian noise is zero, and the discussions are mainly focused on how to determine the SD of Gaussian noise. Here three different ways to choose the standard deviation are considered: SD is fixed in different synaptic weights; SD is proportional to the amplitude of each weight; SD is proportional to the square root of each weight's amplitude (which is equivalent to setting the variance of the Gaussian noise to be proportional to the values of the weights). This study chooses the second method to determine the standard deviation of Gaussian noise. In this situation, weights with high amplitude will have larger noise fluctuations.

Note that the investigated noise in this study is not the random variance in crossbar-based weight matrices which has been widely researched [ZM18, SNP$^+$15]. This random variance originates from the physical properties of materials and the fabrication of transistors therefore this variance is fixed and will not change over time. By contrast, the noise in synaptic weights discussed in this study is the random noise that exists in future high-performance devices such as memristors and magnetic skyrmions at room temperature. This random noise has a certain distribution and its value will change over time.

3.2.4 Related work

Previous research suggests that the biological neural networks of the brain inherently contain noise and rely on the presence of noise to carry out their functions [BBNM11, RT11]. [FSW08] reviewed various noise sources in the nervous system at different levels and showed how noise contributes to trial-to-trial variability. This paper also suggests the potential benefits of noise and illustrates the principles to manage noise. [BCFA01] studies noisy synaptic weights computationally, and shows that the primary noise source of neurons comes from synaptic activities, and the noise in a synapse will reduce to $1/\sqrt{N}$ if N spikes are generated and propagated through this synapse.

In the context of deep spiking neural networks for pattern recognition, the integrate-and-fire mechanism in spiking neurons introduces subthreshold noise and over-threshold noise to the neural network [DNB+15]. Moreover, when using rate coding as an encoding method in SNNs, the input signal is typically noisy as well [PP18]. Even if SNNs inherently contain these noise sources, they could still complete inference with high accuracy in many benchmarks [TGK+19]. Existing research has systematically investigated the impacts of several kinds of noise on the performance of spiking deep belief networks [TGK+19]. However, the inference accuracy of these neural networks is not competitive, and the comparisons in this paper are limited to the same SNNs with different noise levels. By contrast, better results have been achieved by the technique of ANN-to-SNN conversion; meanwhile, ANN-to-SNN conversion allows comparing SNNs to ANNs with the same architecture and synaptic weights.

Though several kinds of noise are investigated in [SNP+15], no study has reported the impact of noisy synaptic weights in SNNs. This gap is filled by this research which studies the tolerance of SNNs to Gaussian noise in synaptic weights. Moreover, the robustness to noisy weights of SNNs and ANNs with the same network architecture are compared, for the first time indicating that SNNs are potentially more robust to noisy weights than ANNs.

3.3 Robustness to noisy weights on MNIST

3.3.1 Network architectures

The experiments are conducted on two kinds of feed-forward neural network: fully connected networks (FCNs) which can be efficiently modeled by a crossbar, and convolutional neural networks (CNNs) which are more powerful models for pattern recognition tasks.

The detailed network parameters are shown in Figure 3.1. In fully connected networks, every two adjacent layers are fully connected in a feed-forward pattern. The output of spiking neurons in the same layer will be sent to all neurons in the subsequent layer without any feedback connection to other layers as shown in Figure 3.1(a). If the connections between two adjacent layers in an FCN become localized and share the same kernels, it will turn into a CNN as shown in Figure 3.1(b). CNNs could achieve better inference results than FCNs with relatively fewer connections, and more local structure information of inputs could be maintained by convolutional kernels. Usually, a pooling layer would be added after a convolutional layer to sub-sample feature maps and reduce the number of parameters. In this study, average pooling layers rather than max-pooling layers are adopted to make the conversion to SNNs easier.

The selected network architectures for evaluation of noise weights are not the state-of-the-art network structure. Impressive results were achieved by using more hidden layers in FCNs [16] and more hidden layers on CNNs [17]. However, to keep the network simple and make it easy to be repeated by other researchers, we use a shallow structure as well as limited optimization techniques for both FCNs and CNNs. Another reason for choosing these relatively shallow network structures is for maintaining the balance between a model and a task. Usually, a bigger deep neural network model is applied only when the tasks to be solved by this model become harder. The task investigated in this study is MNIST, a relatively basic task, which fits the adopted shallow network structure.

3.3.2 ANN-to-SNN conversion

ANN-to-SNN conversion is a relatively successful algorithm to achieve both high inference accuracy and short inference latency in SNNs [DNB$^+$15]. More importantly, it provides an opportunity to compare the performance of ANNs and SNNs in the same network architecture and with the same trained weights. Note that for the standard

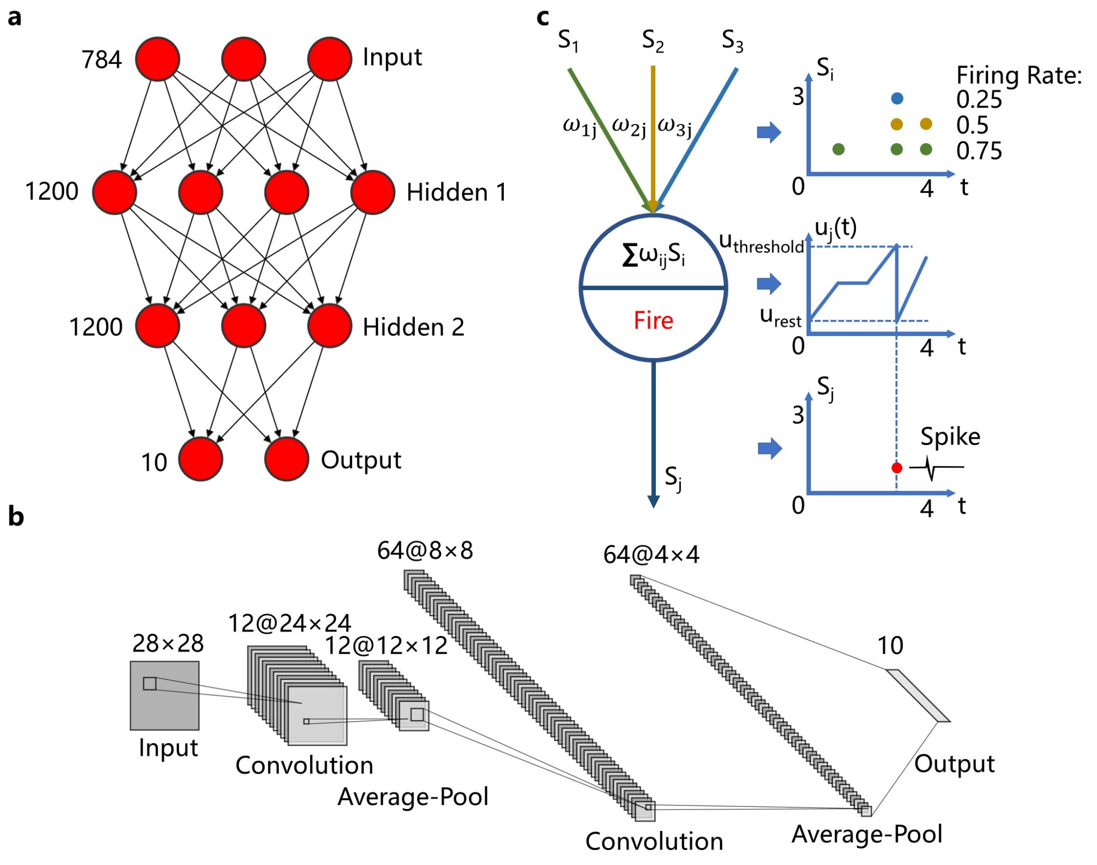

Figure 3.1: The architecture of FCNs and CNNs, and the diagram of the IF neural model

ANN-to-SNN conversion, the weights in ANNs and SNNs are different as the weights are scaled by a factor. In this study, the weights in ANNs and SNNs are kept identical to fairly compare ANNs and SNNs. The thresholds of spiking neurons are scaled instead during ANN-to-SNN conversion. Besides, as the research topic in this study is noisy weights, the biases are deleted to avoid the problem of weight-bias imbalance when using rate coding and the standard integrate-and-fire neurons in SNNs (See Section 4.4 for details).

The weights of the ANN are trained by stochastic gradient descent. After ANN training, the weights are kept unchanged and the analog neurons are replaced with the standard integrate-and-fire neurons. The analog inputs will be encoded as spike trains by rate coding.

The threshold in the layer l is set to D_l/D_{l-1}, whereD_l is the maximum input of analog neurons in the layer l, and D_{l-1} is the maximum input of analog neurons in the previous layer $l-1$. In particular, the threshold in the first hidden layer should be

D_2/D_1. Because D_1 is the maximum input in the input layer and it is equal to 1 in the MNIST dataset, the threshold in the first hidden layer is simplified to D_2, the maximum input in this layer in the ANN.

In Section 3.3.7, we adopt different SNN thresholds to explore the influence of thresholds on inference accuracy and inference latency. The thresholds in these experiments are set by replacing all D_l in the threshold normalization with σD_l. σ is a scale factor. After adding this scale factor, the threshold in the first hidden layer would be σD_2 and the thresholds in other layers are kept unchanged. When σ equals 1, the dynamics of spiking neurons are unchanged. When σ is higher (lower) than 1, spiking neurons will integrate more (fewer) time steps before emitting spikes. The detailed explanations are given below.

Through this weight scaling, the threshold in the first hidden layer will be σ times the maximum input in this layer. According to the theory of data-based normalization, scaling the threshold in a layer will inversely scale the maximum inputs in the subsequent layer (This point is questioned by [SYW+19] because this theory is only strictly correct when the activation in SNNs is exactly the same as that in ANNs. However, this theory is still useful to roughly estimate the change of maximum inputs when the threshold is scaled.). The maximum input in the subsequent layer would be σ times smaller than the original maximum input. Since the threshold in the subsequent layer is unchanged as D_l/D_{l-1}, the threshold in subsequent layers will be σ times their maximum inputs as well. Hence, simply adding σ to D_1 will scale the relative magnitude of the maximum input and the threshold in each layer, which will then scale the firing rate of spiking neurons in each layer.

3.3.3 Metrics for accuracy and latency

The inference result of rate-coded SNNs is the label of the spiking neuron that has the highest firing rate. The output firing rate of SNNs changes over time so that their inference results will change over time [PP18]. These fluctuations of firing rate in the time domain poses a fundamental problem as to when to end SNN simulation and decode the output firing rates as the final inference results.

No paper has been found that discusses the detailed metrics about the choice of simulation time steps on deep SNNs built by ANN-to-SNN conversion. A practical method is provided below:

The first step of the proposed method is to determine a fluctuation range and a monitoring time window. When conducting SNN inference, the SNN simulation will end when the SNN accuracy in the monitoring time window is within the fluctuation range. The SNN latency is the front end of this time window, and the SNN accuracy is the accuracy reached at the front end of this time window. These two parameters can be assigned with different values for different network architectures, tasks and noise levels.

An example to clarify the implementation of this method is given here. Assume that the fluctuation range is set as ±0.05%, and the monitoring time window is 20 time steps. During SNN simulation, the SNN accuracy in any successive 20 time steps is monitored. The accuracy on the first time step of this time window is the baseline accuracy, and the accuraciy in other 19 time steps is compared with this baseline accuracy. Only when the accuracy in the these 19 time steps is all around the baseline accuracy and their differences are within ±0.05%, the simulation is terminated. The inference latency of this SNN simulation is the time point of the beginning of this time window, and the inference accuracy is the accuracy achieved at this time point.

The reason to apply this metric is that accuracy fluctuates widely in rate-coded SNNs, and these fluctuations are enlarged when the synaptic weights in the SNNs are noisy. This can be illustrated from two perspectives. From the perspective of information, the information is accumulated by the integration mechanisms of spiking neurons, and is processed and propagated to the next layer by the firing mechanisms of spiking neurons. At each simulation time step new information comes in to the input layer, and successively propagates to deep layers, eventually leading to the firing rate of the output neurons changing over time and the inference accuracy fluctuating over time. From the perspective of noise, rate-coded SNNs are inherent noisy. Rate-coded SNNs suffer rate coding noise, subthreshold noise, supra threshold noise, and occasional noise (See Chapter 5 for more details.). These noise elements will propagate to deep layers and lead to the SNN output being noisy and the SNN accuracy fluctuating. In the following experiments, the fluctuation range is set as ±0.03%, and the monitoring time window is 10 time steps.

To record latency more precisely for SNNs when injecting high noise levels, an extra metric is introduce. This metric will record the simulation latency when an SNN reaches an acceptable accuracy. This metric is based on the observations that in rate-coded SNNs, increases of accuracy slow down when the accuracy is close to the highest accuracy. Hence, the latency of SNNs to reach slightly lower accuracy is usually obviously shorter than the latency to reach final accuracy, and is less noisy. This lower accuracy is particularly valuable for real-world tasks, as lower accuracy renders lower inference latency and lower power consumption, which are more crucial for many real-world applications.

With this metric, the time when the SNN accuracy is approaching the final accuracy and is within a certain percent accuracy loss from the final inference accuracy will be recorded. For example, when the latency of one percent accuracy loss is selected as the metric and the final inference accuracy is 98.8%, the time point when the inference accuracy is higher than 97.8% for the first time would be recorded. Similarly, the latency of 0.5% accuracy loss is the time point when the accuracy surpasses 98.3% for the first time.

Table 3.1: Training parameters for FCNs and CNNs.

Training parameters	**FCNs**	**CNNs**
Learning rate	1	1
Momentum	0.5	0
Dropout rate	0	0
Training epochs	20	30
Batch size	100	50

3.3.4 Experimental setup

The dataset is MNIST, a handwritten digit database for computer vision and pattern recognition [LBBH98]. It has a training set of 60,000 examples and a testing set of 10,000 examples. These examples are 28*28 pixel pictures of 10 handwritten numbers from 0 to 9. Every pixel has a grey-scale value between 0 and 1.

The FCNs and CNNs are trained in MATLAB using a stochastic gradient descent algorithm. The performance of the ANNs will be slightly different due to different initialized weights and the randomness in the SGD algorithm, thus the final inference accuracy of the ANNs is averaged over 5 trials. The network structure of the FCNs is 784-1200-1200-10 and the network structure of the CNNs is 28x28-12c5-2s-64c5-2s10o as shown in Figure 3.1. The detailed training parameters are shown in Table 3.1. The activation function is ReLU without bias, and it is defined as

$$y_j = max(0, \sum_i w_{ij} y_i), \tag{3.1}$$

where y_j is the output of an analog neuron and y_i is the output of an analog neuron in the previous layer. w_{ij} denotes the synaptic weight connecting neuron i to j. $max(a,b)$

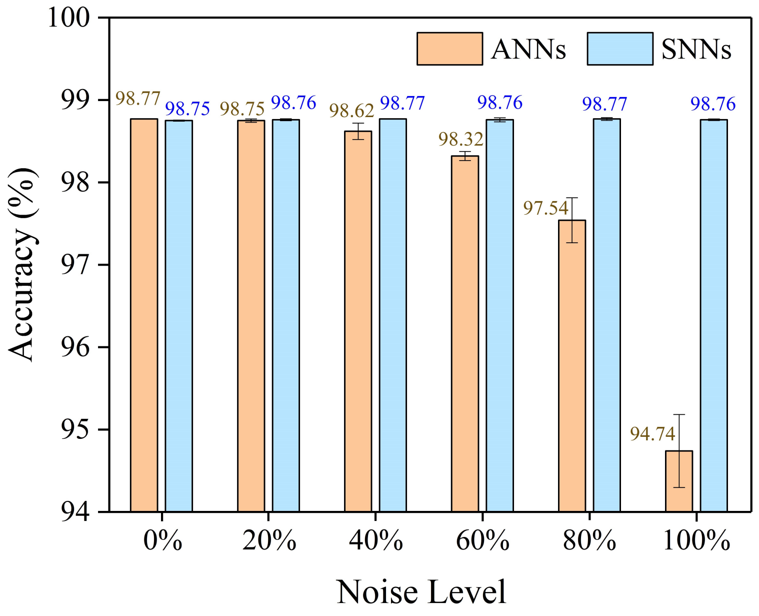

Figure 3.2: Accuracy comparison of the ANNs and the SNNs in the architecture of FCNs for different noise levels.

returns the highest value between *a* and *b*.

After training, all analog neurons are replaced by spiking neurons, and the weights are kept unchanged. The applied spiking neuronal model is the standard IF model. The time resolution of the SNNs is set to 1 ms. The input is changed to rate coding. The maximum pixel intensity has the highest firing rate of 1000 Hz, and the minimum pixel intensity has the firing rate of 0 Hz. The corresponding relationship of firing rate and pixel intensity is linear. The thresholds are set to D_l/D_{l-1}. In the output layer, the inference result is the label of the neuron that has the highest firing rate.

3.3.5 Inference accuracy

The comparison of ANNs and SNNs for different noisy weights on FCNs is shown in Figure 3.2. The X-axis represents the ratio of the Gaussian noise's standard deviation to synaptic weights. The noise level of zero percent on the X-axis represents noise-free synaptic weights. In this figure, the recognition accuracy of the SNNs is slightly lower than that of the ANNs when the noise level is 0%. However, with the increase of noise level, the inference accuracy of the ANNs drops dramatically. When the noise level is 100%, the inference accuracy of the ANNs is only 94.74% on average, and their standard deviation shows an increasing trend for higher noise levels. By contrast, the inference accuracy of the SNNs keeps stable for all noise levels at around 98.76%, and

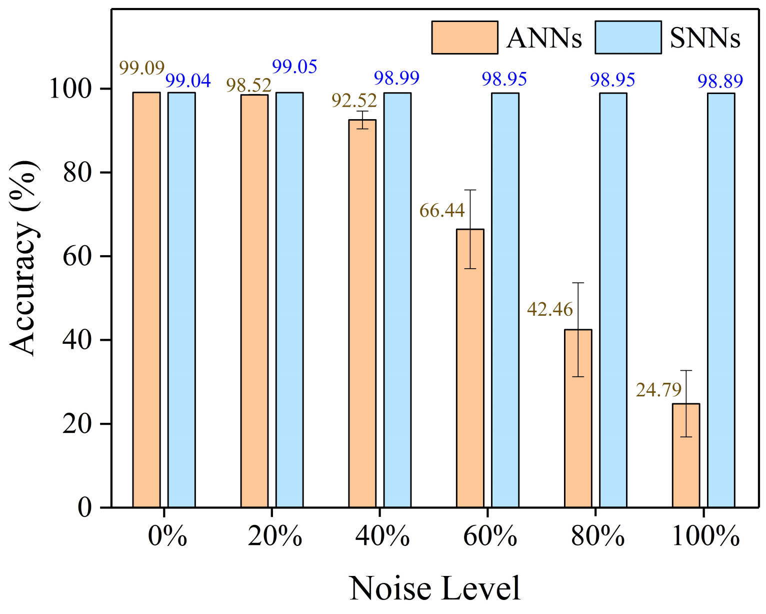

Figure 3.3: Accuracy comparison of the ANNs and the SNNs in the architecture of CNNs for different noise levels.

their standard deviation is smaller than that of ANNs when the noise level is greater than zero percent. When the noise level is 20%, the accuracy of the SNNs has surpassed that of the ANNs.

Figure 3.3 is the comparison of the ANNs and the SNNs for noisy weights on CNNs. The inference accuracy of the ANNs is higher than SNNs when the noise level is 0%. When the noise level is 20%, the accuracy of the ANNs drops below 99%, but the accuracy of the SNNs is still above 99%. The accuracy of the ANNs decreases dramatically and their standard deviation increases to about 10% when the noise level is greater than 40%. By contrast, the accuracy of the SNNs is still 98.89% even when the noise level is 100%. Also, their standard deviation remains small for all noise levels.

3.3.6 Inference latency

The inference accuracy of SNNs represents their inference performance and the effectiveness of ANN-to-SNN conversion, while the inference latency of SNNs reflects their energy efficiency and inference speed. If the inference latency is too long, SNNs will lose their essential advantages of power efficiency and fast inference and will be not suitable for applying to neuromorphic hardware and other cutting-edge devices. The inference latency and standard deviation of spiking FCNs for different noise levels are shown in Table 3.2. As shown in this table, the inference latency increases roughly with the noise level, as does the standard deviation. The inference latency of one percent accuracy loss is relatively stable at around 16 ms and shows minor increases for higher noise levels.

Table 3.2: The inference latency of the spiking FCNs for different noise levels.

Noise level	**Inference latency (ms)**	**Inference latency of 1% loss (ms)**
0%	30.8±4.6	16±0.0
20%	40.2±3.1	16±0.0
40%	41.4±9.8	16±0.0
60%	60.2±26.3	16.8±0.3
80%	69.0±16.4	17.0±0.0
100%	64.8±18.2	17.6±0.9

Table 3.3 illustrates the inference latency of spiking CNNs. As can be seen from this Table, the inference latency of SNNs with a high noise level is significantly longer than that with a low noise level. The inference latency with one percent accuracy loss shows a similar trend. The standard deviation increases for higher noise levels as well.

Table 3.3: The inference latency of the spiking CNNs for different noise levels.

Noise level	**Inference latency (ms)**	**Inference latency of 1% loss (ms)**
0%	206.5±32.5	28±0.0
20%	196.8±60.2	29.6±1.1
40%	222.0±57.2	39.6±1.7
60%	312.0±76.0	71.7±1.7
80%	654.8±168.6	111.4±9.1
100%	701.3±134.8	173.5±11.3

3.3.7 Different thresholds

The thresholds in SNNs will significantly affect their inference accuracy and inference latency. The threshold in this research is set by data-based normalization [DNB$^+$15], in the expectation that this threshold normalization could achieve both fast inference as well as high accuracy. The optimal method to set thresholds may be different when the

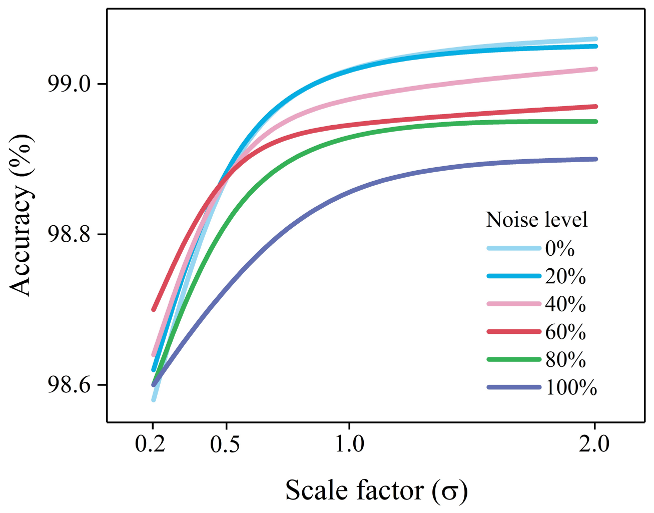

Figure 3.4: The relationship of inference accuracy for different σ.

weights in the SNN are noisy. Driven by that, this section will provide more results on different thresholds.

The thresholds are changed by a scale factor σ as described in Section 3.3.2. Four scale factors σ of 0.2, 0.5, 1, and 2 are tested here. The accuracy of the spiking CNNs for these scale factors and noise levels is shown in Figure 3.4. The accuracy of the spiking FCNs is stable at around 98.77% for all scale factors and noise levels so it will not be presented here. As shown in this figure, the inference accuracy for all noise levels rises with increasing scale factor σ. This indicates that the networks with high thresholds tend to have a higher inference accuracy. As for inference latency, SNNs adopting scale factors of 0.2, 0.5 and 2 show a similar increasing trend for different noise levels as when the scale factor is 1. Figure 3.5 illustrates the relationship of the inference latency ratio to the scale factor σ. This ratio is calculated by dividing the inference latency at 100% noise level by the inference latency at 0% noise level, and it represents how much additional time is needed to cope with Gaussian noise on the weights before SNNs reach the target accuracy. The 1% accuracy loss inference latency is applied as the metric for CNNs and the 0.5% accuracy loss inference latency is applied as the metric for FCNs. The reason to choose 0.5% accuracy loss rather than 1% accuracy loss in FCNs is to improving sensitivity. As seen in Figure 3.5, the network with a higher scale factor has a lower inference latency ratio both in FCNs and CNNs.

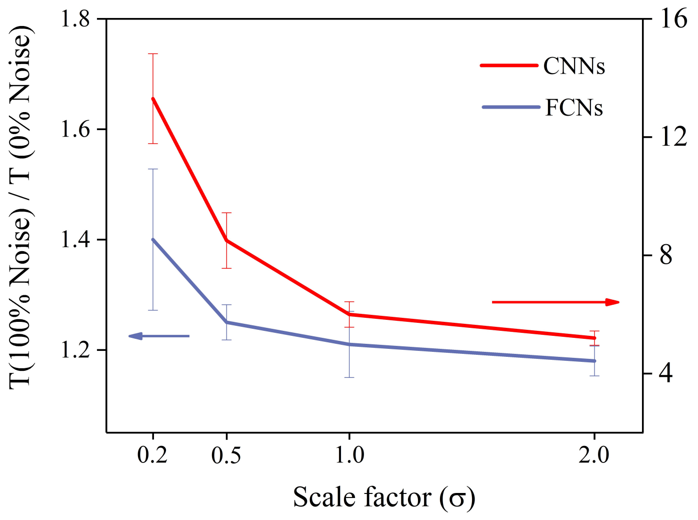

Figure 3.5: The ratio of convergence time of 100% noise and 0% noise on FCNs and CNNs.

3.3.8 Analysis

Inference accuracy

The inference accuracy of ANNs drops considerably with the increase of noise level in synaptic weights. The reason why ANNs are vulnerable to noisy weights is that the noisy weight in ANNs are similar to the weight variance investigated in [SNP+15]. During the inference phase, the noisy weights in ANNs are used once in the feedforward propagation, and the Gaussian noise will only affect the value of synaptic weights once. This Gaussian noise is effectively a random offset which is fixed over time. A high noise level will dramatically change the value of the synaptic weights and affect the function of the ANNs, which causes the final inference accuracy to drop drastically. In addition, a high noise level will introduce more uncertainty in synaptic weights, so the standard deviation of the inference accuracy will increase as well.

By contrast, SNNs have an additional time dimension, and weights with Gaussian noise in SNNs will be used several times during simulation. Thus, the impact of Gaussian noise will be minimized with more spikes transmitted across this synapse. To illustrate it more clearly, assuming a weight with the value w and Gaussian noise $G(0,SD)$, where SD is the standard deviation. The number of spikes transmitted by this synapse is n. Hence, the total effective information transmitted by this synapse is wn, and the total noise is $\sum_n G(0,SD)$. The value of wn will increase with more spikes transmitted. However, the value of $\sum_n G(0,SD)$ equals to $G(0,\sqrt{n}SD)$ according to standard statistics. As a result, the signal-noise ratio $wn/\sum_n G(0,SD)$ will increase with more spikes transmitted.

Inference latency

The accuracy of the SNNs is higher than 98.7% for all noise levels. While SNNs could achieve high inference accuracy at high noise level, their inference latency still goes up at high noise level on both FCN and CNN architectures. This indicates that though SNNs can minimize the impact of noise by averaging noise over time, Gaussian noise on synaptic weights still poses difficulties for SNNs and pushes SNNs to require more time steps to handle the noise .

Influence of Thresholds

In previous sections, why SNNs are more robust than ANNs to noisy weights is explained from a signal-noise ratio perspective. However, the information transmitted by weights will not be fully obtained by postsynaptic neurons due to subthreshold noise and over-threshold noise [DNB$^+$15]. Hence, the choice of thresholds will influence SNNs' robustness to noisy weights. Figures 3.4 and Figure 3.5 illustrated the impact of the scale factor on inference accuracy and inference latency. It can be seen that the robustness to noisy weights is clearly affected by the scale factor on the threshold, and the network with a higher scale factor has better robustness to noisy weights. This is because, in SNNs, the information in weights can only be transmitted to deeper layers through spikes. In the standard integrate-and-fire neuron, the information carried by a spike is always equivalent to the integrated voltage $(u_{threshold} - u_{rest})$ in a spiking neuron no matter how much voltage is cut off by the threshold. $u_{threshold}$ is the threshold of this spiking neuron, and u_{rest} is its resting membrane potential. If the threshold is small, the relative error introduced by this supra-threshold noise will increase. This error will make it difficult for spiking neurons to discriminate the original value of the weights under Gaussian noise. In this situation, the signal-noise ratio may not obviously increase with the number of spikes transmitted through a synapse.

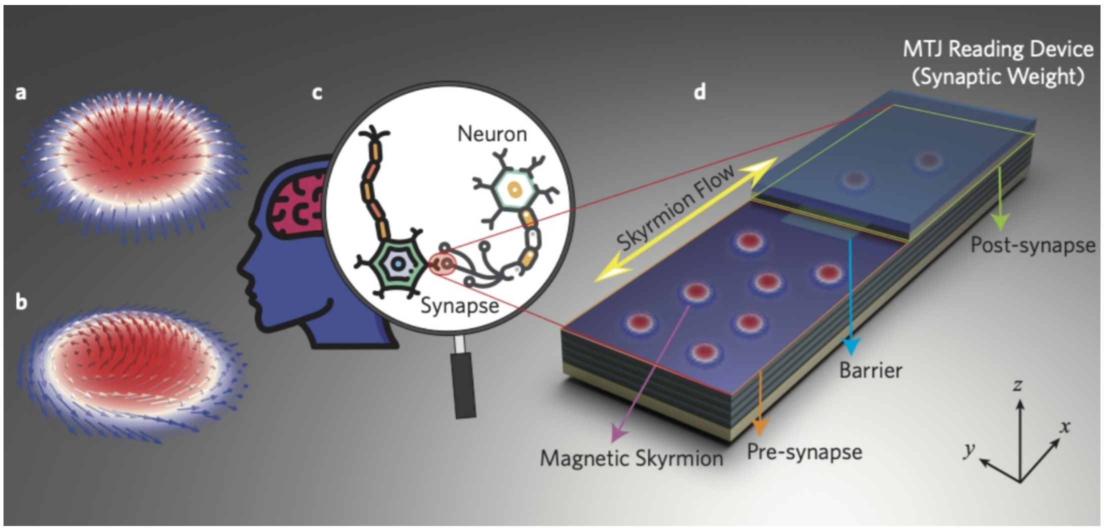

Figure 3.6: The proposed skyrmionic synaptic device. Illustrations of (a) a Néel skyrmion (used in this device) and (b) a Bloch skyrmion spin texture. (c) Schematic of biological neurons connected with a synapse. (d) The proposed nanoscale multilayer skyrmionic synapse device based on skyrmion flow between a pre-synapse and a post-synapse region. The multilayer structure here enables room-temperature operations.

3.4 Optimizing a skyrmion-based SNN

3.4.1 A multi-layer skyrmionic synapse

A multi-layer skyrmionic synapse is proposed and simulated at room temperature. Figure 3.6 shows the schematic diagram of a skyrmionic synapse. As can be seen in this figure, a skyrmionic synapse is composed of three parts: a pre-synapse region, a post-synapse region, and a barrier located in between. The synaptic weight is represented by the conductance of the post-synapse region and measured by a magnetic tunnel junction (MTJ) reading device. This synaptic weight can be tuned by carrying skyrmions to this region, and the movement of skyrmions can be achieved by imposing current pulses on this device. The stability of this proposed device at room temperature is maintained by a multi-layer $[HM_1/FM/HM_2]$ sandwiched structure, where FM represents a ferromagnetic metal layer and HM represents a heavy metal layer.

3.4.2 Towards supervised learning

The existed skyrmion-based SNN simulations usually focus on unsupervised learning, specifically, training SNNs by spike-timing dependent plasticity (STDP). STDP is a biologically-plausible learning rule. Furthermore, the learning process in STDP only involves local information, facilitating on-chip learning on neuromorphic hardware. However, there are two challenges to applying STDP to SNNs.

First, the performance of the SNNs trained by STDP is limited. [DC15] reported an accuracy of 95% on MNIST using a 2-layer SNN, obvious lower than the accuracy of a 2-layer SNN trained by gradient descent which is about 98% on MNIST. The accuracy of SNNs trained by STDP is enhanced in the following works [KGTM18, LPSR18], but the results are still not promising. Also, the scalability of STDP is questionable, as the most STDP-based SNNs are only tested on MNIST and there are rare results demonstrating their effectiveness on more challenging datasets such as ImageNet.

Second, STDP may need huge memory and computing resource during training to get acceptable accuracy. [DC15] shows the impact of the number of neurons in the hidden layer to SNN accuracy. To achieve an accuracy of 82.9% on MNIST, at least 400 excitatory neurons are required; to reach an accuracy of 95%, at least 6400 excitatory neurons are required. On the contrary, an accuracy of 95% can be straightforwardly achieved by using only 100 neurons in the hidden layer in an SNN when adopting supervised learning.

To avoid these problems, this study applies supervised learning to skyrmion-based SNNs, featuring high accuracy during inference and high efficiency during training.

3.4.3 Towards edge inference

As discussed in the previous section, supervised learning shows advantages in accuracy and power efficiency over STDP. Nevertheless, gradient descent and backpropagation, two key techniques in supervised learning are hard to deploy in skyrmionic devices. Gradient descent requires complicated circuits to calculate derivatives and backpropagation needs to access global information instead of local information and to send this information backward. One possible approach to alleviate these high requirements on hardware is applying e-prop, an online learning algorithm without the need for backpropagation [BSS^{+}20]. This research provides a paradigm of developing SNNs offline and on cloud, and deploying them on chips for efficient edge inference.

As shown in Figure 3.7, the proposed edge inference scenario comprises four phases: full-precision on-cloud training, efficient model loading, low-precision edge inference, and data uploading. The on-cloud training is conducted on high performance computing (HPC) clusters with full-precision, where a neural network model is trained by gradient descent and backpropagation on a specific real-world task. The model is then converted to an SNN and loaded to skyrmionic devices at edge. These skyrmionic devices store the weights in the proposed skyrmion-based synapses in low-precision. The proposed skyrmionic synapses are non-volatile and nanoscale, and they can operate at room temperature. These features make them suitable for edge applications under hardware constraints on power, space, and so on.

The edge devices do not need to embed extra circuits on chip for training, bringing more efficient edge inference. When personalized training is necessary among edge devices, the data from users can be collected and sent to cloud devices. This data as well as the whole dataset stored in the cloud is used to fine-tune the model. After the training is completed, the model is reloaded to the edge skyrmionic device.

The main difference between this application scenario and the traditional application scenarios is that the training phase and the inference phase are detached. In edge devices, power can be in short supply, and the resource available for computing and storage can be limited as well. Therefore, inference rather than training is more suitable to be conducted at the edge. On the other hand, the training of the neural network is energy-intensive and relies on high-precision computing and complicated circuits, which can be conducted more efficiently on a fully-digital on-cloud HPC cluster. Some more advanced training techniques such as continuous training and quantization-aware-training which are hard to apply on chip can be applied during on-cloud training. Besides, since this application scenario involves model reloading to edge devices, the efficient weight updating of the proposed skyrmionic synapses is not wasted.

In summary, the presented application scenario forms a closed-loop system, and paves the way for real-world applications of skyrmion-based SNNs.

3.4.4 SNNs with the proposed skyrmionic synapses

This section provides details about applying the proposed skyrmionic synapses to SNNs.

A four-layer fully-connected ANN is trained by stochastic gradient descent. The network architecture is 784-1200-1200-10, which is composed of 784 input neurons, 1200 neurons per hidden layer, and 10 output neurons. Data-based normalization and rate coding are applied to convert the ANN and its inputs to an SNN and input spike trains. In order to deploy room-temperature skyrmionic synapses to deep SNNs, a biology-inspired SNN structure and a weight rescaling technique are proposed.

The original weight matrices of the SNN are W_{21}, W_{32}, and W_{43} which connect the neurons in four layers L_1, L_2, L_3, and L_4 of the SNN. The input is rate coding and is represented as R, which is fed into L_1. This network structure is amended as follows: the numbers of neurons in the input layer and two hidden layers are doubled, forming

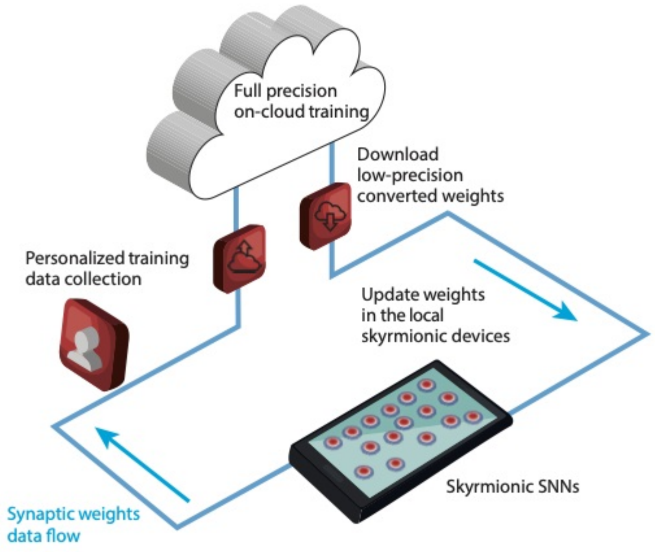
Full precision on-cloud training
Download low-precision converted weights
Personalized training data collection
Update weights in the local skyrmionic devices
Skyrmionic SNNs
Synaptic weights data flow

Figure 3.7: Skyrmionic deep SNNs for edge computing. The proposed skyrmionic deep SNN can be integrated into a framework concept where training takes place in the cloud and updating at the edge. The process is an iterative full-precision training to low-precision conversion cycle: *i*) full precision cloud-based online training and *ii*) skyrmionic devices low-energy updating with offloaded low-precision synaptic weights.

the neurons L_1^+, L_1^-, L_2^+, L_2^-, L_3^+, and L_3^-. Meanwhile, the neurons in the output layer keep unchanged as L_4. The rate coding input spike trains R are duplicated as R^+ and R^-, and fed to L_1^+ and L_1^-, respectively. The weight matrices W_{21}, W_{32}, and W_{43} are split to positive weight matrices W_{21}^+, W_{32}^+, and W_{43}^+ and negative weight matrices W_{21}^-, W_{32}^-, and W_{43}^- with identical size.

Each weight element w in the original weight matrix, e.g. in W_{21}, is represented by the sum of a positive weight element w^+ in W_{21}^+ and a negative weight element w^- in W_{21}^-, which is

$$w = w^+ + w^-. \tag{3.2}$$

w^+ and w^- are within the range of skyrmionic synaptic weights $[1.0, 1.6]$ and $[-1.6, -1.0]$ respectively as shown in Figure 3.8. By summing w^+ and w^- together, w can represent values in the range of $[-0.6, 0.6]$. In other words, the weight ranges of w^+ and w^- are merged to $[-0.6, 0.6]$ by using this method. This proposed network architecture with mirrored neurons and paired synapses is inspired by Dale's principle.

The proposed rescaling method is essentially finding an optimal quantization step size to deal with the challenge of the discrete weight states in skyrmionic synapses.

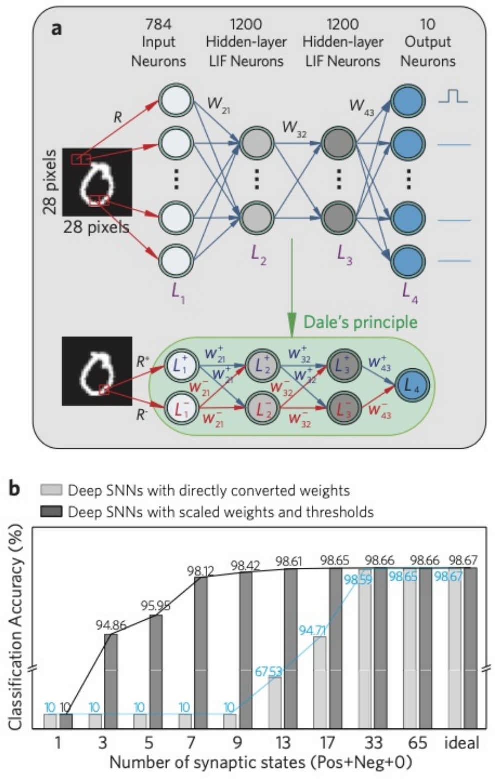

Figure 3.8: The supervised skyrmionic deep SNN. (a) Schematics of the proposed biologically inspired structure of the deep skyrmionic SNN utilizing Dale's principle. (b) Comparison of classification accuracy between i) skyrmionic deep SNNs with directly converted weights and ii) skyrmionic deep SNNs with scaled weights and thresholds, for different number of synaptic states.

The proposal of this method comes from the observation that the accuracy loss when using the conductance of skyrmionic devices to represent weights is due to the weight mismatch. For example, most weighs are distributed in the range between -0.1 and 0.1 after training of the ANN, while the synaptic states at room temperature are in the range between -0.6 and 0.6 after applying the aforementioned SNN structure. This mismatch could cause a huge accuracy loss since most of synaptic states are not utilized. A naive way to improve the accuracy is to apply a scaling factor $\sigma = 0.6/0.1 = 6$ on the threshold of the spiking neurons, which is essentially converting weights from $(-0.1, 0.1)$ to $(-0.6, 0.6)$. In the proposed rescaling approach, the scale factor is obtained by scanning σ and finding the optimal scaling factor that achieves the highest accuracy. This scanning operation on σ is achievable, as with the improving of σ, the accuracy is approximately rising at first and then falling after a certain value. The σ

that enables the neural network to achieve the highest accuracy is the one chosen in this proposed rescaling method. Note that in this method, the neural network accuracy is the only adopted metric to evaluate which scaling factor is optimal. The experimental results show that this is a highly efficient way to find a suitable scaling factor that achieves competitive inference accuracy.

SNNs with different weight precisions, achieved by changing the number of synaptic states to 1, 3, 5, 7, 9, 13, 17, 33, 65, and $+\infty$, are simulated. The number of synaptic states is given by $2X+1$, where X is the number of positive and of negative synaptic weights, and there is one zero weight state. The negative values can be obtained by applying a reverse voltage in the crossbar hardware implementations, and the zero-weight state can be acquired by setting the same value of w^+ and w^-. For example, 13 synaptic states consists of 6 positive, 6 negative and a zero-weight state. The classification accuracy for each weight precision is illustrated in Figure 3.8, where the height difference between the light grey and dark grey columns represents the improvement in accuracy of SNNs with directly converted synaptic weights to SNNs with scaled weights and thresholds. For SNNs with directly converted weights, the results show that the skyrmionic synapse should have 33 synaptic states (16 skyrmionic states) to achieve a $< 1\%$ accuracy loss compared to the ideal full-precision synapses. In comparison, SNNs with scaled weights and thresholds show a much faster increase of classification accuracy when the number of synaptic states increases. The accuracy exceeds $\sim 98\%$ at only 7 synaptic states (3 skyrmionic synaptic states). Notably, we obtain a superior $\sim 98.61\%$ classification accuracy with 13 synaptic states (6 skyrmionic states of RT skyrmionic synapse), which is merely $\sim 0.06\%$ lower than the SNNs with ideal full-precision synapses. The results here demonstrate the excellent potential for the use of the proposed skyrmionic synapses in neuromorphic computing, especially when deployed in deep SNNs and ensuring room-temperature operations.

3.5 Deploying on SpiNNaker

A four-layer fully-connected SNN whose network architecture is 784-1200-1200-10 is deployed on SpiNNaker to valid its function on a real neuromorphic machine. The weights in this SNN are static instead of noisy, since SpiNNaker is a digital neuromorphic platform and it can store and represent weights reliably.

The goal of this research is demonstrating the process to deploy an offline-trained functional SNN to a neuromorphic machine. Before this study, there was no demonstration of SNNs built by deep learning techniques on SpiNNaker. Deep learning techniques in this context narrowly refer to deep ANNs that are trained by supervised learning and backpopagation.

A deep belief network trained by the Restricted Boltzmann Machine methods was applied to SpiNNaker, achieving an accuracy of 95.01% [SNG+15]. Concurrent research [PSRGSGLB20] applies a convolutional SNN to SpiNNaker, yet its accuracy is limited as well (98.20% on MNIST) and the tackled challenges are different from this research. The deployment of a convolutional SNN on SpiNNaker faces the challenge of the efficient representation of convolutional kernels on SpiNNaker. This research, in contrast, uses a fully-connected neural network architecture so will not meet this problem. However, the number of synapses and synaptic events in this fully-connected SNN are orders of magnitude greater than that in the convolutional SNN, posing a challenge of efficient resource management of the low-power neuromorphic hardware.

3.5.1 SpiNNaker

SpiNNaker [FGTP14, FLP+12] is a fully digital neuromorphic hardware composed of one million ARM cores, with the aim of achieving large-scale brain modeling in biological real-time. "SpiNNaker" is a crude contraction of "spiking neural network architecture". It supports massively-parallel, large-scale neural computation with highly-efficient interconnection between the processing units which is inspired by the connectivity characteristics of the mammalian brain. The tremendous inter-neuron communications is effectively managed by an asynchronous packet-switching multicast spike routing network. Another salient feature of SpiNNaker is flexibility, as it offers various neuronal models and learning rules to choose for neural modeling.

In SpiNNaker, each chip contains 18 ARM968 processing cores, and each core is associated with an adjacent memory including 32KB for instructions (instruction tightly coupled memory, ITCM) and 64KB for data storage (data tightly coupled memory, DTCM). These 18 cores share an 128-Mbyte off-chip low-power mobile dual-data-rate (DDR) SDRAM (Synchronous Dynamic Random Access Memory) where most of the information on synaptic connectivity is held.

The spike routing network in SpiNNaker is optimized to deliver very large numbers of very small packets. Each packet carries a single "spike" event whose length is 40 bits: 32 bits for an AER (address event representation) identifier and 8 bits for management. This identifier contains information about the neuron that spiked. In addition

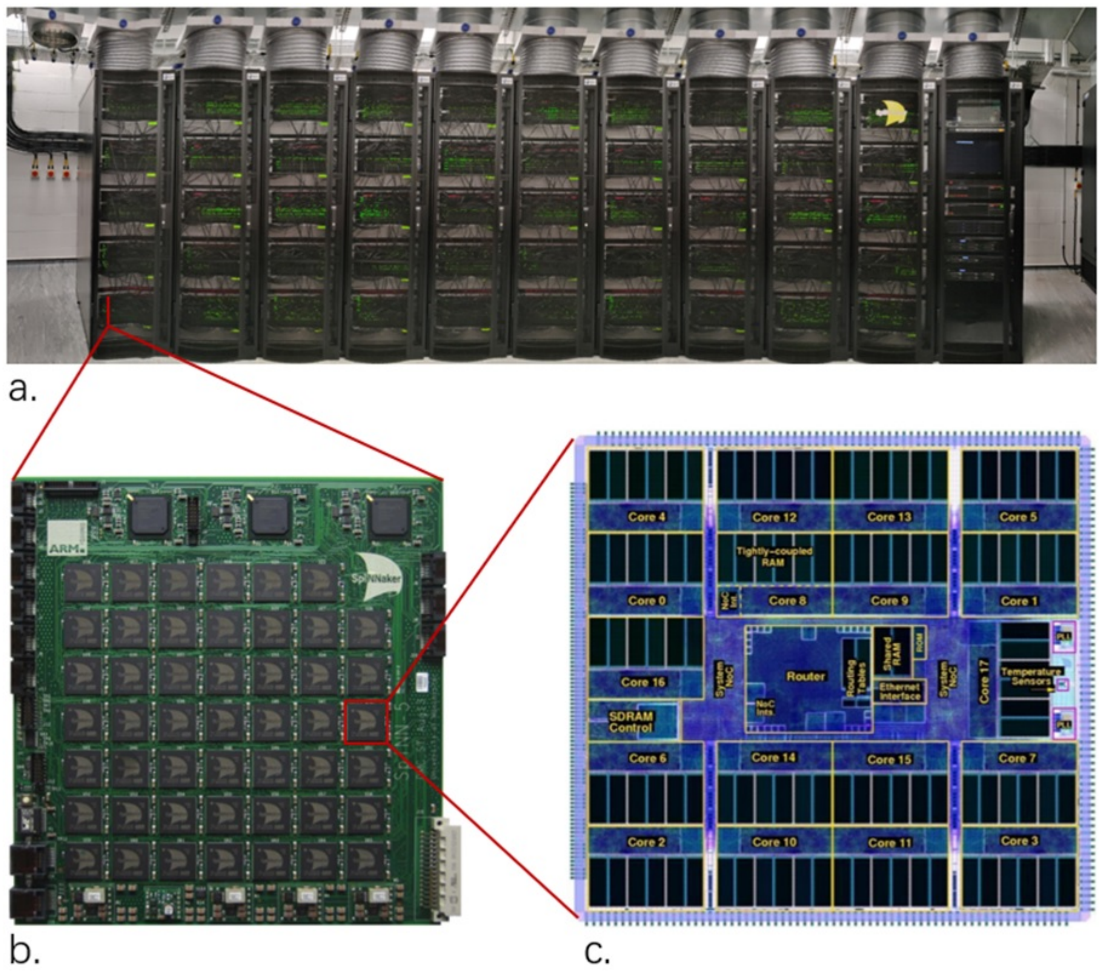

Figure 3.9: Different scales of SpiNNaker neuromorphic hardware. (a). SpiNNaker big machine with 1 million cores. (b). 48-node board with 864 cores. (c). A plot of a single chip with 18 cores.

to AER "spikes", SpiNNaker also supports non-AER information flows through the same communication mechanism. Figure 3.9 shows the different scales of SpiNNaker neuromorphic hardware.

3.5.2 Implementation details

The first step in deploying the offline-trained SNN is defining the network structure and parameters on SpiNNaker. The fully-connected SNN comprises 4 layers so there are four groups of neurons to be built on sPyNNaker, a PyNN-based software package that enables simulating SNNs on SpiNNaker [RBB+18]. The neuronal model is leaky integrate-and-fire, with the time constant of 700 ms (while the time resolution during the

SNN simulation on SpiNNaker is 1 ms), a comparably slow leak rate, and the refractory time of 1 ms. Specially, there is one extra neuron connected to all these neurons through one-to-all connections to reset these neurons for new input samples. How this works will be introduced latter.

The synaptic weights of this SNN are converted manually from MATLAB to SpiNNaker: The weight matrices are extracted from MATLAB files and are converted to lists in Python. Each item in these lists includes the index of a presynaptic neuron, the index of a postsynaptic neuron, and the weight value of the synapse connecting these two neurons. These lists are used to build the synaptic connections of the SNN on SpiNNaker. Note that the excitatory synapses and inhibitory synapses need to be built separately, so each weight matrix is divided to two lists, one for positive weights and one for negative weights. The synaptic model is conductance-based, and its time constant is 0.1 ms, which means that the current is injected from synapses to postsynaptic neurons instantly.

The samples in the MNIST dataset are converted to rate-coded spike trains and fed to the SNN as inputs. The maximum input firing rate is 1000 Hz and the encoding time window is 100 ms. At the end of this time window, the aforementioned resetting neuron will generate a spike to all other neurons. The synaptic weights of this one-to-all connection is set as a very high positive value, so the generated spike will trigger all postsynaptic neurons to fire and their membrane potential can be reset consequently. After the membrane potentials of all neurons in this SNN have been cleaned by this mechanism, the next sample will be sent to the network for inference.

3.5.3 Results

The recognition accuracy of the SNN on SpiNNaker is 98.63% on test dataset, with only 0.01% ANN-to-SNN conversion loss compared with the original ANN in MATLAB. 98.63% is the highest reported accuracy of MNIST on SpiNNaker to date, compared to other reported accuracies of 95.01% [SNG$^+$15] and 98.2% [PSRGSGLB20]. Also, 0.01% conversion loss is lower than other reported conversion loss of 1.05% [SNG$^+$15] and 0.76% [PSRGSGLB20].

The information transmission in this SNN is sparse. Typically, there are only 6 to 7 neurons firing in these layers on each simulated time step. This low firing rate brings comparably few synaptic events, alleviating the burden on the memory buffer and computing resource during SNN simulation. Further reductions of the firing rate can be achieved by either optimizing the firing rate during training or applying a higher neuronal threshold during ANN-to-SNN conversion.

Another feature of this SNN is lower connectivity. A naive weight pruning technique is applied to remove small weights. After weight pruning, 55% of the weights in this SNN are removed. Weight pruning can directly benefit to SNN simulation on SpiNNaker: First, SpiNNaker represent weights by lists so the removed weights will be removed from these lists, which brings reduced memory to store these weight lists. Second, after weight pruning, a generated spike in a presynaptic neuron will only needs to be sent to postsynaptic neurons whose synapse is not deleted, leading to reduced spike communications.

Compared with the standard IF model used in the original MATLAB SNN simulation [DNB+15], the adopted neuronal model in the SNN built on SpiNNaker has leak and refractory times but still achieves a competitive accuracy. It should be a small step to reconcile biological plausibility and computational performance in SNNs.

3.6 Summary

Research in SNNs is still in the early stages and a substantial amount of the research is driven by exploring the differences between SNNs and their counterpart ANNs. Few practical advantages of SNNs have been found up to now. The present study on noisy weights provided the first comprehensive assessment of the effect of Gaussian noise on synaptic weights and found that SNNs are significantly more robust to Gaussian noise on synaptic weights than conventional ANNs. This robustness to noise comes from the time dimension of SNNs and the integrate-and-fire mechanism of spiking neurons, which assists in our understanding of the characteristics of SNNs. This study paves the way for the adoption of a spike-based computational paradigm on cutting-edge devices such as memristors and magnetic skyrmions for noise-resilient edge AI inference.

The non-idealities such as weight variations and noisy weights, have been researched for years in neuromorphic hardware and neural network accelerators. However, studies of noisy weights from the perspective of SNN algorithms are lacking. The presented research on noisy weights filled this gap. Also, the robustness to noisy weights investigated in this research provided a new metric for SNN algorithms. Note that these SNN algorithms are not necessarily designed for noisy memory devices such as memristors and skyrmions. Instead, the evaluation of noise robustness from the perspective of SNN algorithms has independent research value. Most of the current SNN algorithms are only benchmarked on inference accuracy and inference latency, and state-of-the-art performance is considered as the primary goal. This research highlighted the significance of noise resilience especially the robustness to noise on weights. By evaluating SNN algorithms with these extended metrics, more SNN algorithms can be evaluated instead of solely the one with state-of-the-art accuracy and latency.

There are a variety of ways to evaluate SNN algorithms on this new proposed metric. The robustness to noisy weights of a SNN algorithm can be compared to the baseline ANN algorithm, or other related SNN algorithms, or the same SNN algorithm before applying a certain optimization technique. This study exemplified the first category. When deploying intelligence at the edge for real-world tasks, algorithm-hardware co-design has been proven to be a practical approach to improve performance and robustness. Though this research solely investigated the noise robustness of algorithms, further optimizations under noise can be applied to SNN algorithms to prepare for future implementations on noisy hardware in advance.

Section 3.4 described how to embed the skyrmionic synapses into SNNs for a pattern recognition task. To fully exploit the limited-precision weights and the intrinsic merits of RT MML skyrmionic synapses, the skyrmionic synapse was integrated into a deep SNN that is trained by supervised learning. A high classification accuracy of about 98.61% was achieved with 13 weight states obtained from the proposed skyrmionic synapses. The simulation of deep SNNs with the proposed skyrmionic synapses enabled wider possibilities for energy-efficient hardware implementations to perform neuromorphic computing. Also, this study provided a application paradigm of edge inference and shed new light on real-world applications of skyrmion-based SNNs.

The contents in Section 3.5 demonstrated the deployment of a biologically-plausible neural network model on a biologically-inspired neuromorphic hardware SpiNNaker. It has achieved state-of-the-art accuracy, and features spatial-temporal sparsity.

Chapter 4

Biological plausibility of SNNs

4.1 Introduction

Initially, SNNs were applied to describe the dynamics of biological neural systems and neural circuits [ESC$^+$12, MJS07] by computational neuroscientists, benefiting from their biological plausibility and computational capability. With these successes in biological neural modeling and promoted by the parallel achievements of deep learning, there are various approaches proposed to build deep SNNs to achieve biologically-plausible, event-based computing [CCK15, DNB$^+$15, SNP$^+$15, HE15, EAM$^+$15, LDP16, OW16, NAPD17].

The most successful approach to date trains an ANN using supervised learning and converts the trained ANN into a rate-based SNN [CCK15, HE15, LCF17]. Lossless conversion can be achieved by this method compared with direct training, but it usually requires using spiking neuron models that are "artificially" modified to transmit precise information to the subsequent layer [RLH$^+$17, SM19, SM20]. These methods can achieve good accuracy on many benchmarks but work against the original objective of using spiking neurons in deep neural networks. In other word, current deep SNN research is trading biological plausibility to achieving competetive network performance.

Another desirable way to go to build functional SNNs is applying more biologically-plausible spiking neuron models. Though there will be many obstacles to this research direction, this is an unavoidable process if we wish to unlock the mysteries of the human brain and duplicate them on the machine. Previous work has explored building deep SNNs using the standard IF model [SYW$^+$19], but the modeling of bias and batch normalization remains unaddressed until now. These two elements directly affect the inference accuracy in ANNs which is also the target accuracy of SNNs. The lack of these elements is obstructing the performance of SNNs built by the standard IF model as well as their further application.

This research demonstrates that it is possible to retain both high performance and biological plausibility in SNNs. Also, some insights into managing the firing rate range in deep SNNs are offered. The main contributions of this study are:

- State-of-the-art accuracy on the MNIST and CIFAR-10 data sets is achieved compared to other research that builds deep SNNs using the standard IF model.
- Bias and batch normalization are modeled in deep SNNs whilst retaining biological fidelity.
- The presented method features a 2.5 to 7.5 times lower spike rate than previous state-of-the-art ANN-to-SNN conversion methods, so paves the way to run these networks on neuromorphic hardware.
- The significance of input normalization during ANN-to-SNN conversion is emphasized and integrated into the proposed normalization approach, MCR-Norm (an abbreviation of minimum chain rule normalization), to form the first systematic parameter normalization strategy for SNNs.

4.2 Related work

This section provides a brief review of studies directly related to the proposed normalization approach. Several successful parameter normalization approaches have been proposed in deep SNNs to facilitate ANN-to-SNN conversion [DNB$^+$15, RLH$^+$17, SYW$^+$19].

Data-based normalization scales weights according to the input patterns of spiking neurons. The input patterns of the test data set are estimated from the input patterns of the training data set. This estimation relies on an independent and identically distributed (IID) hypothesis between the training data set and the test data set.

Percentile data-based normalization provides an additional bias normalization equation on the basis of data-based normalization [RLH$^+$17]. The balance of weights and biases in SNNs is maintained by adding an extra membrane potential reset mechanism to the IF model. The outliers of the neuron inputs are discarded to improve firing rates. However, this discard may bring accuracy loss if the outliers contain important information.

Spike-Norm calculates the maximum inputs in spiking neurons layer-by-layer in a sufficiently long time window to determine the values of the thresholds [SYW$^+$19]. Spike-Norm successfully applies the original IF model to challenging data sets, but does not include the normalization of biases and batch normalization layers. Due to the inherent noise in SNNs, the normalized thresholds calculated by this method vary in different trials and with different lengths of time-window.

4.3 Current methods and gaps

In the beginning, ANN-to-SNN conversion is applied to a standard IF model, and its effectiveness is demonstrated on the MNIST dataset [DNB$^+$15]. However, the standard IF model fails to scale to more challenging datasets such as CIFAR-10 and ImageNet, and deeper network structures such as VGG-16 and ResNet. Some methods are proposed to tackle this problem, and the following are the two typical solutions as well as their pros and cons:

- Modifying the reset mechanism of the standard IF model and replaying rate coding to analog coding, to transmit more precise information layer-by-layer in SNNs. This solution provides a method to achieve lossless ANN-to-SNN conversion. Also, this method supports the modeling of bias and batch normalization, and enables SNNs to be run with high firing rates, which brings higher network capacity, higher accuracy, and lower latency. However, biological plausibility is damaged.

- Sticking to the standard IF model and rate coding to keep biological plausibility, and modifying the network structure to delete bias and batch normalization, two problematic elements which are hard to model by the standard IF model. This solution can maintain more biological plausibility, with the compromise of lower SNN performance. Specifically, the ANN accuracy drops due to the lack of bias and batch normalization, and ANN-to-SNN conversion loss increases due to using a more noisy neuronal model and more noisy inputs in the SNN. Another impact is that the SNNs built by this method are required to run at low firing rates to reduce the noise in the standard IF model, which leads to a prolonged latency of the SNNs.

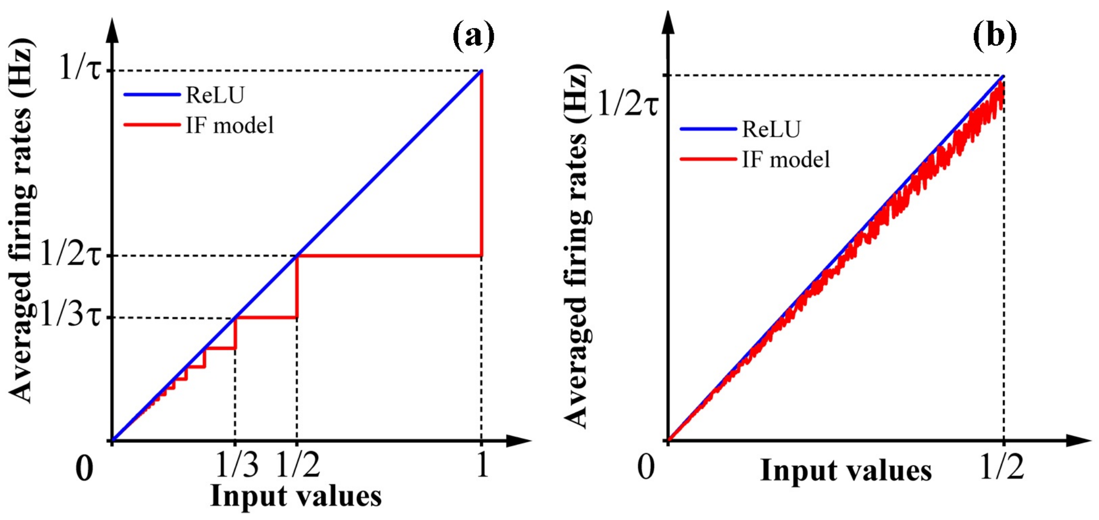

Figure 4.1: The response curves of the IF model and ReLU. The input is a constant current injection, and the synapse model is the delta model. (a) The response curve of the IF model and ReLU. (b) Changing the input to a statistical current injection. τ is the time resolution during the SNN simulation. When τ is 1 ms, $1/\tau$ is 1000 Hz. Other firing rates can be calculated in the same way.

Currently, Network performance is the primary consideration in many deep SNN studies, and the first solution is dominant. The second solution is limited due to long latency, lower accuracy and incapacity to model the biases in the ANNs. This study focus on improving the second solution to enable effective modeling of biases as well as improving accuracy.

4.4 Firing rate degeneration and weight-bias imbalance

This section will introduce a “firing rate degeneration” phenomenon in deep SNNs when building SNNs using the standard IF model and rate coding after ANN-to-SNN conversion. This phenomenon may bring the risk of too long latency, and moreover, it will lead to the mismatch between weights and bias (which is named as weight-bias imbalance in this chapter) so prevent the modeling of bias and batch normalization in an SNN.

Figure 4.1(a) illustrates the response curve of the standard IF model to constant current injection and a comparison with the target response curve, e.g. ReLU, in ANNs. It can be seen from this figure that the shape of the IF response curve is unevenly stepped and a gap exists between these two curves.

The stepped shape can almost be eliminated when the input is noisy. The noisy

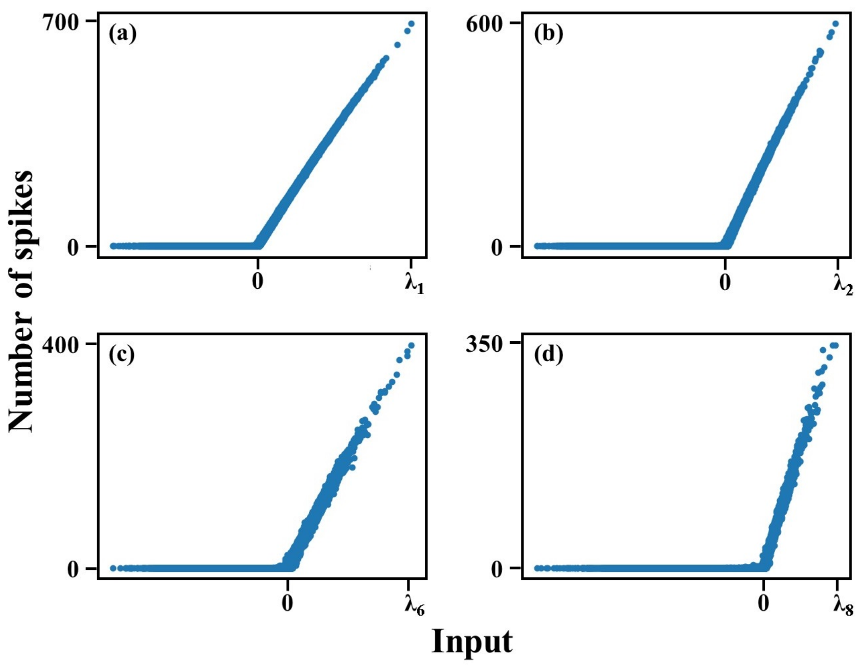

Figure 4.2: The firing rate degeneration phenomenon on deep SNNs. The figures with labels (a) to (d) show the response curves in layers 1, 2, 6 and 8 respectively. The neural model is the standard IF model and the weights are normalized by the traditional parameter normalization method [RLH+17].

input can be generated by changing the constant current injection to a Gaussian current injection. In deep SNNs, the input of a spiking neuron is noisy as well since it receives randomly generated spikes from spiking neurons in the previous layer. A typical response curve of the IF model smoothed by noisy input is shown in Figure 4.1(b). As shown in this figure, the response curve of the IF model is now more similar to ReLU, especially when the input value is small. However, the output firing rate is still lower than the expected ReLU-like shape; this is called the "firing rate degeneration" phenomenon in the IF model. This phenomenon imposes two challenges to deep SNNs that are converted from ANNs:

(1). The degenerated firing rate in one layer generates a ripple effect in subsequent layers. More specifically, with the network going deeper, the spike firing rate range gradually moves towards a lower region. (Figure 4.2). When the network is deep, the spiking neurons in deep layers may be rarely activated so require more time steps to transmit the same amount of information, which eventually leads to a long inference latency.

(2). When conducting the standard data-based normalization (equations are shown in Equation 2.33 and Equation 2.34), weights need to be scaled additionally by a factor λ_{n-1}, the maximum activation in the previous layer, to compensate for the weight scaling in the previous layer. Nevertheless, since the response curve of the spiking neuron suffers from firing rate degeneration (as well as its ripple effect in subsequent layers), the correct scaling factor needs to be adjusted correspondingly to compensate for the degenerated input firing rate in the previous layer. Without this adjustment, the weights and biases will be mismatched. Also, since the firing rate degeneration is more severe in deeper layers, the weight-bias imbalance is worse when SNNs have more layers, which leads to huge accuracy loss in SNNs. In other words, weight-bias imbalance prevents SNNs from scaling to deeper neural network models..

4.5 Proposed methods

To overcome these challenges caused by firing rate degeneration, and to achieve high-performance deep SNNs using the standard IF model, a normalization method called MCR-Norm, an abbreviation of minimum chain rule normalization, is proposed. The "minimum" in this term emphasizes that the firing rate range in each layer after applying MCR-Norm is maintained identically at a low level. "chain rule" emphasizes that the normalization is performed layer-by-layer under a chain rule.

The core idea of MCR-Norm is that the accuracy of deep SNNs can be kept by scaling weights additionally to compensate for the firing rate degeneration and regulating the firing rate to a predefined range. MCR-Norm is modified on the data-based normalization, whose equations are shown below:

$$\widetilde{w}_n = w_n * \frac{\lambda_{n-1}}{\lambda_n}, \tag{4.1}$$

$$\widetilde{b}_n = b_n * \frac{1}{\lambda_n}. \tag{4.2}$$

w_n and b_n ($\widetilde{w}_n$ and $\widetilde{b}_n$) represent the weights and biases before (after) the normalization in layer n respectively. λ_n and λ_{n-1} are the maximum activations in layer n and layer $n-1$ respectively. The subscript of these scale factors represents the layer in which the scaling calculation happens. After conducting parameter normalization by these equations, an ANN is converted to an SNN whose firing rate range in each layer is (0 Hz, 1000 Hz) if time resolution is 1 ms.

If targeting to build an SNNs with a lower firing rate range, a scale factor g_n can be added to λ_n in the parameter normalization equations. After replacing λ_n with λ_n/g_n as well as replacing λ_{n-1} with λ_{n-1}/g_{n-1}, the equations become

$$\widetilde{w_n} = w_n * \frac{\lambda_{n-1} \cdot g_n}{\lambda_n \cdot g_{n-1}}, \tag{4.3}$$

$$\widetilde{b_n} = b_n * \frac{g_n}{\lambda_n}. \tag{4.4}$$

The value of g_n is determined by the target firing rate range. For example, if the target firing rate range is (0 Hz, 200 Hz), g_n will be 0.2. Note that these equations do not consider the impact of the firing rate degeneration phenomenon and assume the firing rate range is the same in each layer, e.g. (0 Hz, 200 Hz). In reality, SNNs suffer firing rate degeneration and the firing rate range is different in each layer. This suggests the need of further regulations on g_n to control the firing rate in each layer within the predefined target range.

In MCR-Norm, the following equations are adopted to normalize network parameters:

$$\widetilde{w_n} = w_n * \frac{\lambda_{n-1} \cdot g_n}{\lambda_n \cdot g_{n-1}}, \tag{4.5}$$

$$\widetilde{b_n} = b_n * \frac{h_{n-1} \cdot g_n}{\lambda_n \cdot g_{n-1}}. \tag{4.6}$$

The role of g_n in Equation 4.5 and 4.6 is compensating the impact of firing rate degeneration and regulating the firing rate in each layer to a predefined range (e.g., 0 Hz to 200 Hz). g_n is calculated layer-wise: In each layer, its response curve with different g_n is plotted, and g_n that makes the response curve closest to the target firing rate range is chosen. More details about calculating g_n and its computational complexity is presented in the following section.

Another contribution of MCR-Norm is a method to achieve the correct scaling of biases. As shown in Equation 4.6, the calculation of bias normalization is different from Equation 4.4 that does not consider firing rate degeneration. h_{n-1} is a predefined factor that is determined by the target firing rate. For example, if the target firing rate range is (0 Hz, 200 Hz), the value of h_{n-1} will be 0.2.

The correctness of the equation of bias normalization in MCR-Norm can be checked in the following way: According to Equation 4.5 and 4.6, the weights are normalized additionally by a factor λ_{n-1}/h_{n-1}. This additional scaling compensates for the mapping in the previous layer $n-1$ from the ANNs activation in the range of [0, λ_{n-1}] to the SNN firing rates in the range of [0, $h_{n-1} \cdot 1000Hz$].

4.5.1 Efficient calculation of g_n

The most computational complexity of MCR-Norm comes from the calculation of g_n. It is not feasible to scan all possible g_n to obtain the desired firing rate range. To reduce its computational complexity, two techniques are applied:

(1). Since the scale factor g_n is positively related to the upper bound of the firing rate range in layer n, some techniques such as dichotomy can be used to reduce the computational complexity to get the target firing rate range. After using dichotomy, the deep SNN needs an average of a few thousand time steps to get the correct scale factor g_n in each layer. Then it is around the same computational complexity as Spike-Norm.

(2) The scanning operation is conducted on the valid set rather than on the whole training set. This technique can make MCR-Norm a few orders of magnitude faster. Also, using the scale factors derived from the calibration set is proven to work well on the test set.

4.5.2 Input normalization

In addition to normalizing the weights and biases inside the network, the normalization of the inputs fed into the network also has a major impact on the performance of the SNN. Previous SNN normalization techniques skip the input normalization and start the normalization from the first hidden layer [DNB$^+$15, RLH$^+$17, SYW$^+$19]. The correctness of these normalization methods relies on the assumption that the inputs to the ANN are in the range (0, 1) or (-1, 1), and the normalized SNN inputs are in the range of (0, $1/\tau$), which is (0, 1000 Hz) if the time resolution τ is 1 ms. These two assumptions are not always fulfilled, e.g. the inputs of CIFAR-10 after input normalization in ANNs are in the range about (-2.2, 2.2), and the input firing rate range of SNNs can be (0, 400 Hz) in some configurations [DNB$^+$15].

To correct this fault in parameter normalization after ANN-to-SNN conversion, and to enable the flexible configuration of input firing rates, MCR-Norm is extended to include normalization in the input layer. This is achieved by considering that the input range (0, λ_0) in the ANN is linearly mapped to the input spike firing rate range (0,g_0/τ) in the SNN. Thus, the MCR-Norm equation starts in the input layer rather than the first hidden layer. g_0 controls the target firing rate range, e.g. g_0 is set to 0.4 if the target input firing rate range is (0, 400 Hz). h_0 is simply set as 1. The normalization of negative inputs is achieved in the same way by using signed spikes [RLH$^+$17].

4.5.3 Further improvements and compatibility

Some experimental results in this study show that fine-tuning h_n can achieve better accuracy. This may be because the response curve of real spiking neurons is not exactly linear even when the firing rate range is low. Due to this reason, the predefined h_n may not be the optimal value but need a small amount of fine-tuning around.

The idea behind the MCR-Norm to manage the firing rate range layer-wise can be applied to other neuron models as well. When the neuron model is the IF model with an extra reset mechanism and the input is analog, g_n will be predefined and have the same value as h_n, without the need to conduct scanning and fine-tuning.

4.6 Considerations behind the proposed methods

4.6.1 Inspirations

The firing behaviour of spiking neuron can be controlled by one hyper-parameter threshold: a high threshold causes a low output firing rate and a low threshold results in a high output firing rate. MCR-Norm adjusts weights (which is inversely equivalent to changing thresholds) slightly to compensate for the effect of firing rate degeneration in the IF neuron. Then the firing rate range is identical in every layer, and biases can be normalized according to this determined firing rate range.

The phenomenon of firing rate degeneration occurs in every layer so the weights (thresholds) need to be tuned in each layer. Inspired by batch normalization in ANNs which normalizes the inputs in each layer to the same distribution, MCR-Norm is conducted layer-wise and the spike firing rate is normalized to the same range.

One main goal of neuromorphic hardware is conducting the simulation of biological neural networks. The biological neural network is naturally sparse and the maximum firing rate is below 200 Hz [Len03], rather than 1000 Hz as used in most SNN simulations. From the hardware viewpoint, the construction of neuromorphic hardware usually considers the worst situation during running. For example, the highest spike rate that might be received by one hardware node, and whether this traffic upper bound is the same in different hardware nodes. This consideration of the highest firing rate is reflected in the proposed MCR-Norm, where the firing rate upper bound is limited to an identical low value in every layer. Consequently, when the simulated results are applied to the real neuromorphic hardware, the spike traffic upper bound is low and identical in hardware nodes.

4.6.2 Normalizing weights vs normalizing biases

To maintain the balance between weights and biases under the firing rate degeneration phenomenon, there are two feasible solutions: scaling weights to match biases, or scaling biases to match weights. The latter solution can achieve a balance between parameters but cannot solve the firing rate degeneration problem. With that in mind, weight scaling is chosen in the present method.

4.6.3 Low firing rates vs high firing rates

MCR-Norm scales weights to make the firing rate range in every layer identical. This section quantitatively explores the optimal firing rate range to achieve high performance. Under the same experimental conditions (listed at the end of this section), the upper bound of the firing rate range is adjusted and its impact on the network performance, specifically, inference accuracy and inference latency are recorded as shown in Figure 4.3. It can be seen that the accuracy drops when the firing rate is high, while high latency appears at low firing rates. The results suggest that unlike modified IF models, there is a trade-off inside the choice of the working zone of the standard IF model. The balance point is approximately between 200 Hz - 400 Hz. This is in line with some biological observations that neurons usually fire below 200 Hz as the cost of spike generation is high [Len03].

A toy model was built to explore the optimal identical firing rate range. The data set is CIFAR-10. The network architecture is shown in Table 4.1. This ANN model only has weights and does not apply biases and batch normalization, to avoid the problem of weight-bias mismatch after conversion to SNN. After ANN-to-SNN conversion, the SNNs are built using the standard IF model with a time resolution of 1 ms and are normalized to different firing rate ranges as shown in Figure 4.3. The baseline accuracy is 89.43%.

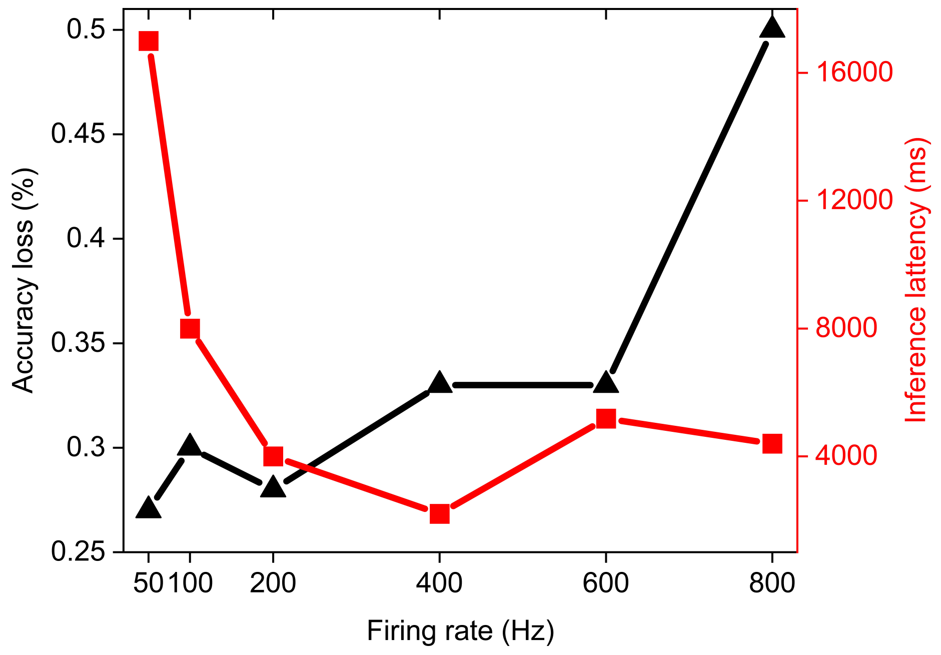

Figure 4.3: Inference accuracy loss after ANN-to-SNN conversion and inference latency for different firing rates. The data set is CIFAR-10 [RLH+17].

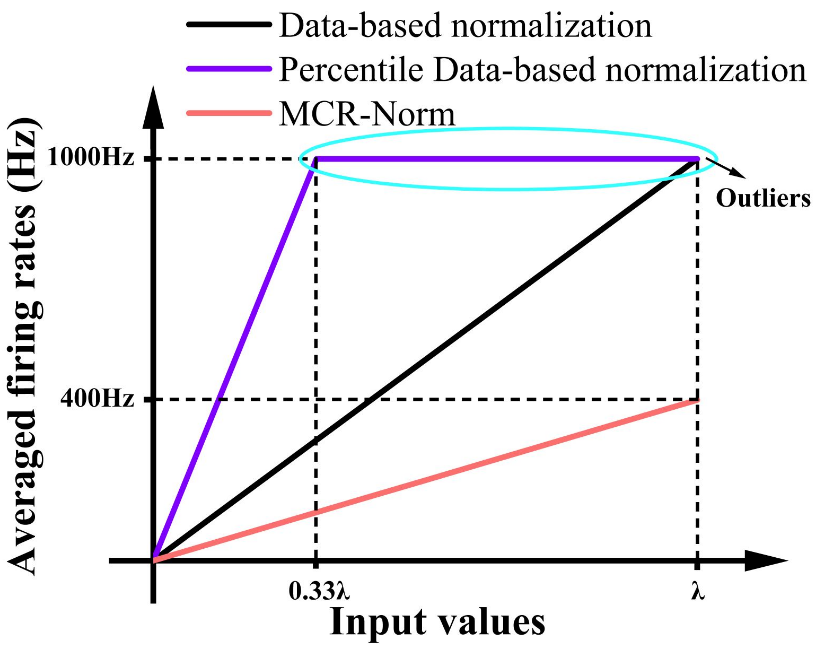

Figure 4.4: The response curves of spiking neurons under different parameter normalization methods.

Table 4.1: Network structures for MNIST and CIFAR-10

MNIST
28*28-64c3BN-128 c3BN-128 c3BN-p2-D0.1 -128 c3BN-256c3BN-256c3BN-p2-D0.1-256 c3BN -512 c3BN-D0.1 -2048FC-D0.4-10FC
CIFAR-10
32*32*3-64c3BN-128c3BN-128c3BN-p2-D0.1 -128 c3BN-256c3BN-p2-D0.1-256 c3BN- 256c3BN -p2-D0.1-512 c3BN-p2-D0.1-2048FC-D0.4-10FC

4.6.4 Comparison to other normalization methods

Figure 4.4 compares the response curves of spiking neurons in MCR-Norm and other parameter normalization strategies. All these approaches normalize thresholds proportional to the maximum activation λ. Note that this diagram is simplified to show the essence of these parameter normalization methods. The real response curve of spiking neurons has variation and noise. It shows that for the same inputs, the spike firing rate after using MCR-Norm is 2.5 times smaller than data-based normalization and 7.5 times smaller than percentile data-based normalization (Ignoring the 0.1% to 1% outlier neurons).

4.7 Benchmarks

4.7.1 Experimental setup

The datasets used here are MNIST and CIFAR-10 [LBBH98, HRFS16]. The network architectures are inspired by earlier work [HRFS16] and are given in Table 4.1. There are several convolutional layers with BN layers to extract features, and some fully-connected (FC) layers at the end. The size of the convolution kernels is 3*3 and the size of the pooling kernels is 2*2. Spatial dropout is applied after some pooling layers and the standard dropout is applied before the FC layers. The activation function used in these networks is ReLU. The network parameters and hyper-parameters are shown in Table 4.2.

The spiking neuron model used here is the standard IF model and the synapse model is the delta model [DNB$^+$15]. A statistical rate coding is applied to map input values to averaged firing rates. The time resolution is 1 ms.

Table 4.2: Training parameters and hyperparameters of neural networks for MNIST and CIFAR-10.

Dataset	MNIST	CIFAR-10
Criterion	Cross entropy	Cross entropy
Optimizer	Adadelta	Adadelta
Epochs	150	150
Batch size	100	100
Other techniques	Early stopping	Early stopping, data augmentation, L2 regulator

All experiments are run on a computer with a Core i7 2.8 GHz multi-core CPU, 8 GB RAM, and a GTX1050Ti GPU. The operating system on this computer is Windows10. The ANNs and SNNs are built using Pytorch.

4.7.2 Inference accuracy

MCR-Norm was tested on the MNIST and CIFAR-10 pattern recognition data sets. The inference accuracy is compared with various ANN-to-SNN conversion techniques in Table 4.3. The proposed method achieved a state-of-the-art accuracy of 99.71% on MNIST, benefiting from the powerful architecture and the use of the batch normalization technique. More importantly, it achieved zero accuracy loss after ANN-to-SNN conversion, without any modifications to the standard IF model.

For CIFAR-10, it achieved an accuracy of 93.60%, which is the best reported accuracy with the standard IF model up to now. It shows approximately 2% accuracy improvement over SNNs built using the standard IF model without biases and BN layers [SYW$^+$19]. The ANN-to-SNN conversion accuracy loss is 0.29%, which is better than many methods using less biologically plausible neural models [RLH$^+$17, SM19, SM20]; however, this conversion accuracy loss is higher than [RLH$^+$17, HSR20] which uses the IF model with an extra membrane potential reset mechanism.

4.7.3 Inference latency

The SNN takes 100 ms to converge to the final accuracy on MNIST during inference. The inference latency of the SNN on CIFAR-10 is much longer than on MNIST. It takes

Table 4.3: Accuracy loss on MNIST and CIFAR-10 with ANN-to-SNN conversion techniques.

Neuron type	**Bias**	**BN**	**Dataset**	**ANN Loss (%)**	**SNN Loss(%)**
Original IF model [DNB+15]	No	No	MNIST	0.86	0.90
IF model with subtraction mechanism [RLH+17]	Yes	Yes	MNIST	0.56	0.56
Standard IF model (this work)	Yes	Yes	MNIST	0.29	0.29
IF model with subtraction mechanism [RLH+17]	Yes	Yes	CIFAR-10	8.09	9.15
Original IF model [SYW+19]	No	No	CIFAR-10	8.3	8.45
IF model with soft reset [HSR20]	No	No	CIFAR-10	6.37	6.37
AMOS unit [SM19]	Yes	Yes	CIFAR-10	7.07	7.58
FS neuron [SM20]	Yes	Yes	CIFAR-10	7.01	7.58
Standard IF model (this work)	Yes	Yes	CIFAR-10	6.11	6.40

1,700 ms to reach the final accuracy as shown in Figure 4.5. This result on CIFAR-10 is compared with a deep SNN converted from the same ANN model but normalized using data-based normalization and applied the IF model with subtraction mechanism [RLH+17]. The results are shown in Figure 4.5 and it shows that the proposed method does not incur additional latency during inference. What is more, the firing rate range is (0, 400 Hz) rather than (0, 1000 Hz) used in data-based normalization and percentile data-based normalization.

4.8 Summary

This research proposed the "MCR-Norm" normalization method to tune parameters systematically after ANN-to-SNN conversion and, using that, it achieved competitive accuracy on the MNIST and CIFAR-10 pattern recognition data sets with the standard IF model. As well as the absolute accuracy of the deep SNN, the accuracy loss compared with the original ANN after MCR-Norm is crucial to represent the efficiency of ANN-to-SNN conversion. Thanks to the layer-wise optimization, the reported accuracy loss was only 0.29% on CIFAR-10 and 0% on MNIST.

The core of MCR-Norm is to control the firing rate range at a low level, rather than allowing varying and/or high firing rates. This study showed that a standard IF model naturally prefers a firing rate range close to the one used by biological neurons. The proposed normalization approach also paved the way to efficiently adopt SNNs on neuromorphic hardware [FGTP14].

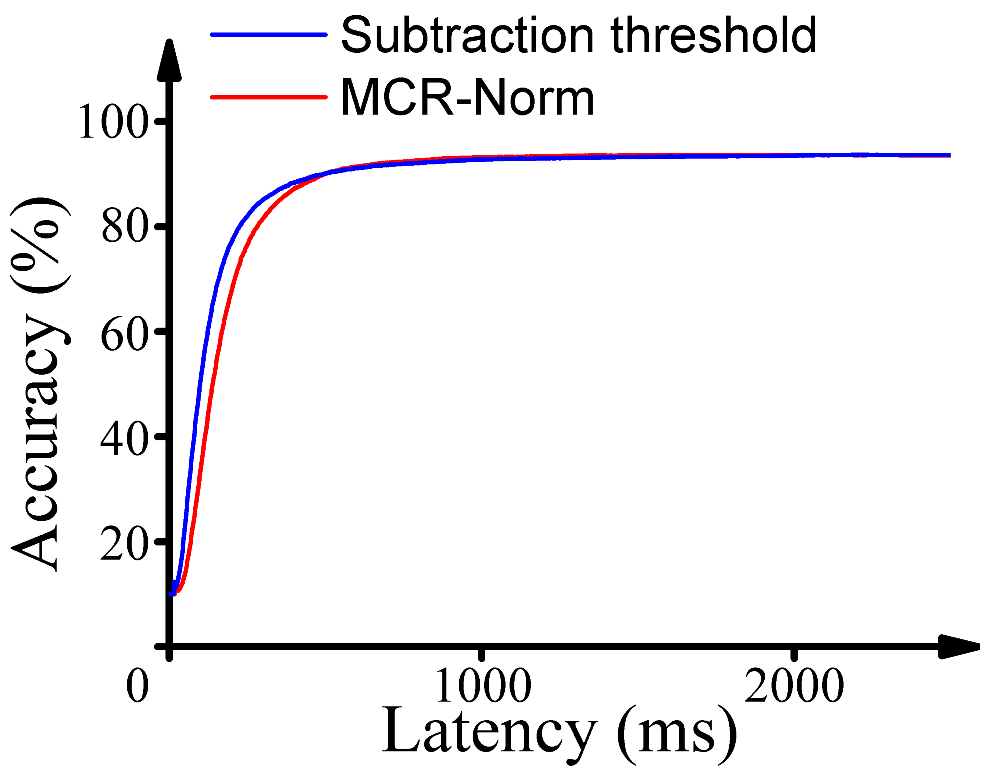

Figure 4.5: The convergence time of the SNN on CIFAR-10.

Chapter 5

A quantization framework for fast SNNs

5.1 Introduction

The performance of SNNs, and specifically their inference accuracy, has improved significantly over recent years, driven by the motivation to prove that SNNs are as functional as their ANN counterparts. Emerging techniques show that lossless SNN accuracy is possible [DNB$^+$15, RLH$^+$17, SYW$^+$19, LZZ21, DG21].

With this success in the pursuit of inference accuracy, considerable scholarly attention has shifted to the aspect of inference latency[DG21, HCO$^+$21, HC20, LDD$^+$21]. This research topic is referred to as "latency optimization in SNNs", or simply as "fast SNNs". Fast SNNs are achieved either by conducting more efficient ANN-to-SNN conversion or by training the SNNs directly by surrogate gradients. Nevertheless, with the reduction in latency comes degradation in accuracy, resulting in the well-known accuracy-latency trade-off in SNNs.

The main goal of this research is to build fast SNNs while avoiding accuracy loss. In particular, This research chooses an ANN-to-SNN conversion technique to minimize accuracy loss while applying a novel quantization framework to push latency below ten time steps, for the first time. Thus, a highly-effective method to build state-of-the-art ultra-fast, high-accuracy SNNs is demonstrated. The key contributions of this study are listed below.

- **Performance**: The proposed method overcomes the accuracy loss problems previously seen in fast SNNs after conversion from ANNs, and achieves a state-of-the-art accuracy and latency. Specifically, on ImageNet it achieves the accuracy of 70.18% in 8 time steps and 74.36% in 10 times steps.

- **Information compression**: The fast SNNs are generated by compressing activation precision. This research discusses how to achieve extremely low-bit activation compression (down to 2 bits) and, more importantly, how to ensure the compatibility of this technique with SNNs.
- **Noise suppression**: a new type of noise in spiking neurons, which is called "occasional noise", is identified, and shown that it is the main obstacle to achieving competitive accuracy for fast SNNs. An effective approach is proposed to suppress its negative effect on SNN performance.
- **Framework**: A comprehensive quantization framework for fast SNNs (QFFS) is proposed to include the proposed information compression, noise suppression techniques, and other techniques. This framework enables SNNs to be built with both high inference accuracy and low inference latency. Beyond that, further improvements in accuracy, latency, and biological plausibility are possible based on this framework.

5.2 Related work

The inference latency of SNNs has continued to reduce over the last five years. The early demonstration of SNNs on ImageNet needed about 2000 time steps to get competitive accuracy [SYW$^+$19]. Rueckauer et al [RLH$^+$17] applied a modified integrate-and-fire model and analog input to facilitate the accuracy and latency of SNNs. The 0.1% to 1% activation outliers were discarded to further reduce the SNN latency.

The modified IF model and analog input were then widely used in SNN research, and there was a surge of interest in further shortening the inference latency of the SNNs. Hwang et al [HCO$^+$21] and Deng et al [DG21] used a pre-charged membrane potential and bias shift respectively to eliminate systematic error during ANN-to-SNN conversion. Another contribution of [DG21] is they used clipped ReLU during ANN training to match the response curve of SNNs better. Ho et al [HC20] applied clipped ReLU during ANN training as well, but the clipping point in each layer is trainable. Through these efforts, ANN-to-SNN conversion has become increasingly effective, and the inference latency of the SNNs has reduced to about 30 time steps. However, the accuracy loss after conversion is considerable when the latency is pushed down towards ten time steps.

There are two parallel methods have been proposed for achieving fast SNNs: direct training [FYC$^+$21a] and tandem learning [WXH$^+$21]. Nevertheless, both of these two methods are hindered by the huge memory budget during training and accuracy degeneration during inference. The comparison between the proposed method in this study and these two methods is illustrated in Section 5.6.

The research into ANN quantization is extraordinarily prosperous, pursuing low computation and memory budgets for deploying Tiny Machine Learning applications on edge devices [WS19]. The standard methods are post-training quantization and quantization-aware training [Kri18]. Using these two approaches, most neural network models can achieve lossless accuracy with 8-bit precision compared with the corresponding full precision models. Further reduction of the precision mainly relies on modifying gradients [EMB$^+$19]. The extreme situation is using 1-bit weights and activations to conduct inference, an approach called binary neural networks (BNNs) [QGL$^+$20].

Research into applying quantization to SNNs is comparably limited. Quantization techniques are primarily adopted to compress the model footprint of the SNNs, and these techniques have been applied to weights, neuronal parameters, and neuronal state to deploy SNN algorithms on memory-constrained neuromorphic hardware [LN21, SJ20, CGR21].

Meanwhile, several studies have explored the effectiveness of using quantization techniques to promote fast SNNs [WXH$^+$21, MHAK21, BFD$^+$21]. However, these methods either fail to scale to ImageNet, or suffer severe accuracy degradation. The main challenge for fast SNNs - preventing accuracy drop when pushing down the inference latency - has not been dealt with. The main differences between the study presented in this chapter and these studies are:

- This research provides a comprehensive analysis of occasional noise and provides a corresponding noise suppression method, which is shown to be crucial to achieving competitive accuracy for SNNs within strictly limited time steps.
- This research enables building SNNs with 2-bit precision and loss-less accuracy (while other research uses 4-bit to 8-bit quantization and suffers serious accuracy loss). Also, some key modifications to the standard quantization techniques are emphasized in this paper to better fit the dynamics of spiking neurons.

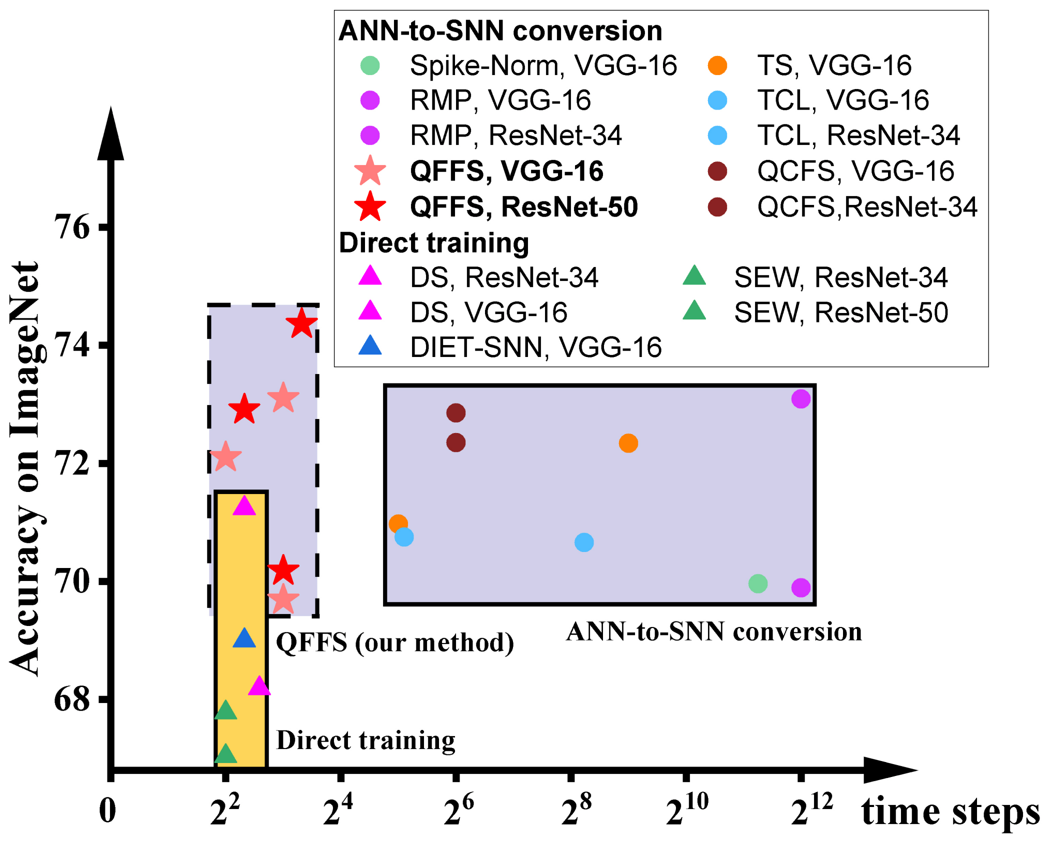

Figure 5.1: The accuracy and latency of SNNs built by different methods (Spike-Norm [SYW+19], TS [DG21]), RMP [HSR20], TCL [HC20], QCFS [BFD+21] DS [LGZ+21], SEW [FYC+21a], DIET-SNN [RR20], and QFFS proposed in this paper. ANN-to-SNN conversion delivers high accuracy, and direct training delivers low latency. The proposed QFFS approach pushes the latency, when using ANN-to-SNN conversion, to a level similar to that using direct training. Also, our method shows about 2.5% higher accuracy than the best accuracy achieved by direct training.

- Other methods usually apply analog neurons instead of spiking neurons in the output layer of the SNNs, to improve the resolution of the output layer so keeping competitive accuracy on tasks such as ImageNet. The presented method shows the possibility of achieving an accuracy higher than 70% on ImageNet with spiking neurons in the output layer.

5.3 Motivation

The motivation for this research is to develop a practical method to reconcile the accuracy-latency trade-off in SNNs. Currently, there are two dominant methods to build

SNNs: ANN-to-SNN conversion, and direct training with surrogate gradients. Which method is chosen depends on the requirements of the SNNs under different application scenarios. Generally, ANN-to-SNN conversion features high accuracy, while direct training features low latency. In other words, an accuracy-latency trade-off arises with current SNNs. To illustrate this, the accuracy and latency of SNNs using these two methods are shown in Figure 5.1, grouped by different symbols. It is obvious that these two methods are currently distinguished from each other and have distinct working zones. To date, there is no approach to building SNNs with latency matching that using direct training and accuracy equivalent to that using ANN-to-SNN conversion. The purpose of this paper is to push the bounds of both of these methods, to promote the reconciliation of the accuracy-latency trade-off in SNNs. Specifically, this research focuses on improving the latency of ANN-to-SNN conversion, for the first time to a level close to that of direct training. As a result, the latency gap between these two methods can be closed, while ANN-to-SNN conversion will show clear advantages in training (lower memory budget and shorter training time) and in inference (higher accuracy) than direct training.

5.4 Materials and methods

Considering a rate-coded SNN built using the ANN-to-SNN conversion technique [DNB+15, RLH+17], the accuracy of the SNN, $Acc(SNN)$, is given by

$$Acc(SNN) = Acc(ANN) - Loss(conversion), \tag{5.1}$$

where $Acc(ANN)$ is the accuracy of the full precision ANN, $Acc(SNN)$ is the accuracy of the SNN, and how close this is to the full ANN accuracy depends on $Loss(conversio$ $n)$, which is the accuracy loss introduced by ANN-to-SNN conversion. If we quantize the ANN prior to conversion then this becomes

$$Acc(SNN) = Acc(Quant\ ANN) - Loss(conversion), \tag{5.2}$$

where $Acc(Quant\ ANN)$ is the accuracy of the quantized ANN. After this modification, the target accuracy of the SNN becomes $Acc(Quant\ ANN)$. Benefitting from the recent advances in ANN quantization techniques, $Acc(Quant\ ANN)$ is increasingly close to $Acc(ANN)$ even when the bit precision is strictly constrained. Hence, the baseline accuracy of the SNN, $Acc(Quant\ ANN)$, is maintained in principle.

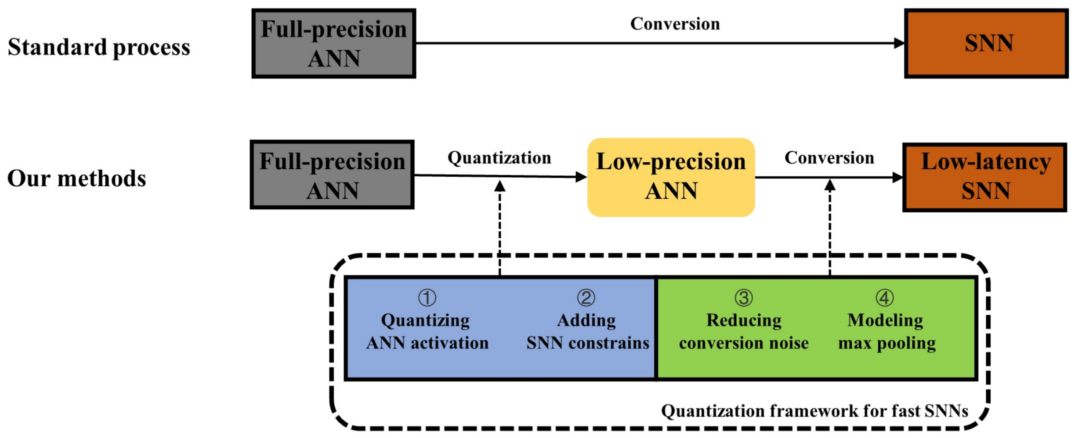

Figure 5.2: The general ANN-to-SNN conversion diagram and the proposed approach to achieve fast SNNs.

Minimising the ANN-to-SNN conversion loss $Loss(conversion)$ is crucial to ensuring that the SNN approaches this baseline accuracy. It is achieved by analyzing the neuronal dynamics of spiking neurons to find the origin of the accuracy loss and eliminate it (Sections 5.4.2 and 5.4.3).

As for the inference latency, this study empirically show that the inference latency of the SNN and the activation bit-width of the ANN are correlated after ANN-to-SNN conversion, so a fast SNN can be built by using a quantized ANN. This is covered in the following section, describing the applied ANN quantization techniques and highlighting the modifications to the standard quantization techniques to ensure compatibility with SNNs.

Another issue addressed in this study is the simulation of max pooling in SNNs. This study proposes a practical SNN max pooling method to improve the accuracy of the SNN compared with that using average pooling, without compromising its event-based nature.

The exploration of these aspects forms a quantization framework for fast SNNs (QFFS) as shown in Figure 5.2. How this framework delivers further improvements in SNN performance is discussed at the end of the Chapter. The detailed equations related to the ANN-to-SNN conversion and Quant-ANN-to-SNN conversion are provided in Section 5.4.5.

5.4.1 Information compression during training

Implementing quantization training

The inference accuracy of SNNs increases with the simulation time step but, in essence, it increases with the amount of information transmitted by uniform spike trains. After sufficient information has been accumulated, a comparatively reliable "decision" can be made, and this point in time is defined as the inference latency of the SNN. For example, to transmit 8-bit information, at least 255 ms is required in a rate-coded SNN with a 1ms time resolution. If temporal coding is used, the required length of time is also related to the target information bit-width. Thus, reducing the required information bit-width is the key to achieving fast SNNs.

When using an ANN-to-SNN conversion technique, the required bit precision of the SNN is determined by the activation bit precision of the ANN. So the problem becomes that of building a quantized ANN with an activation bit-width as low as possible while maintaining high accuracy.

During the last decade, ANN quantization techniques have been at the centre of much attention, and the standard quantization methods (post-training quantization and quantization-aware training) have increasingly matured. For instance, there are well-developed and easily accessed APIs in PyTorch to conduct ANN quantization by these two methods. Though these APIs are easy to access, these two standard methods are not suitable for this research, as they fail to achieve competitive accuracy in extremely low bit precision such as 2 bits. Hence, the first obstacle is to choose a more effective quantization method than the standard post-training quantization and quantization-aware training. This obstacle is also part of the reasons why early attempts to use ANN quantization to promote fast SNNs either failed to scale to challenging datasets (such as ImageNet) or suffered high accuracy loss in fast SNNs (e.g. 6% on ImageNet).

The ANN quantization technique chosen in this study is based on LSQ [EMB$^+$19]. LSQ defines the gradients of the quantization step size to prevent activations from being too close to quantization transmission points. It can enable network quantization down to 2 bits while minimizing the accuracy loss introduced by quantization. This quantization accuracy loss equals $Acc(ANN) - Acc(Quant\ ANN)$, which is the accuracy difference between the full-precision ANN model and the quantized-ANN model. According to the original LSQ paper, this accuracy loss is about 3.2% for 2-bit ANN and 1.1% for 3-bit ANN on ResNet-50. How to reduce this accuracy loss is discussed in Section 5.7.

Table 5.1: The main differences between general quantization techniques and the quantization techniques for fast SNNs.

	Standard quantization	Quantization for fast SNNs
Position	Network input, network output, ReLU, arithmetic	ReLU
Procedure	Fake quantization training - convert to integer model - run on the backend with integer arithmetic accelerator	Fake quantization training
Operation	Rounding	Grounding
Granularity	Per tensor, per channel	Per tensor
Benefits	Speed up inference and reduce memory budget	Reduce inference latency

Additionally, LSQ is not open-sourced, and the implementation of LSQ in this study has not achieved a similar accuracy to that claimed in that paper. For example, the 2-bit and 3-bit ANN quantization results on ImageNet are 1.5% and 2.3% lower than the reported results in the LSQ paper respectively. This suggests that there is further scope for improving the presented methods.

Modifications to promote compatibility with SNNs

As the quantized ANN will be converted into an SNN in the future, the ANN quantization technique should be compatible with the properties of SNNs.

Table 5.1 lists the modifications to the general ANN quantization technique. Only the activation quantization is applied during training and the input, weights, and biases are left as floating-point. In the standard quantization procedure, the model generated by fake quantization training will be converted to an integer model and run on different backends. Here, only the fake quantization training is applied and the model is then converted to an SNN. The granularity of activation quantization is the tensor. There are two modifications found crucial to the final SNN performance:

Firstly, many quantization techniques including the applied LSQ leave the output layer in floating-point to render better accuracy, e.g. 4% higher on ImageNet than that quantizing the output layer. However, modeling floating-point with spiking neurons is expensive. It needs either many time steps to generate enough spikes to reach the same precision, or to use integrate-but-not-fire neurons in the SNN and represent the high-precision information by the neurons' membrane potentials. These two solutions will damage the inference latency or biological plausibility of the built SNNs. In this research, the sensitivity of SNN performance to the activation precision in the output layer is investigated. The results support the choice of an optimal precision to promote competitive accuracy and latency.

Secondly, the integrate-and-fire mechanism in spiking neurons corresponds to rounding down rather than rounding to nearest which is generally used in ANN quantization. This issue can be solved either by shifting to using the rounding down during the quantization training, or sticking to using rounding to the nearest during quantization and compensating for it later. Considering that the quantization method is fine-tuning an already trained full-precision model, rounding down during quantization will introduce a systematic error resulting in a considerable accuracy loss, especially for low-precision quantization. This study sticks to using rounding to nearest during quantization and compensates for it in the SNN by pre-charging the membrane potential [HCO+21].

5.4.2 Occasional noise

The types of noise causing accuracy loss during ANN-to-SNN conversion are summarized for the first time in [DNB+15], where three kinds of noise – sub-threshold noise, supra-threshold noise, and rate-coding noise - are illustrated. Some research categorizes these as errors rather than noise. This study sticks to calling them noise, to emphasize their randomness and uncertainty.

This research argues that there is a fourth kind of noise, which is called as occasional noise. Occasional noise refers to the phenomenon that occasional spikes are generated in spiking neurons where they should not be. For example, consider an artificial neuron with an input of 0.4, and the threshold of the corresponding spiking neuron is 1 with a simulation period of 10 time steps. During this simulation period, the average input value is 0.4 and the neuron should generate 4 spikes in 10 time steps, in the simplest situation. However, the inputs of a spiking neuron are weighted spikes, with random spike timing. Possible situations include:

- **Situation 1:** The input of this spiking neuron is -1 in the first 9 time steps and 13 in the last time step. Here the number of spikes generated is 1 instead of 4.
- **Situation 2:** The input is 1 in the first 8 time steps and -2 in the last 2 time steps. Then the summed input is 8*1 + 2*(-2) = 4, which is correct, but the generated spike count is 8 instead of 4.

These erroneously generated spikes will propagate through the network and cause an accuracy drop in the SNN. To verify this, Some ANNs are trained with different

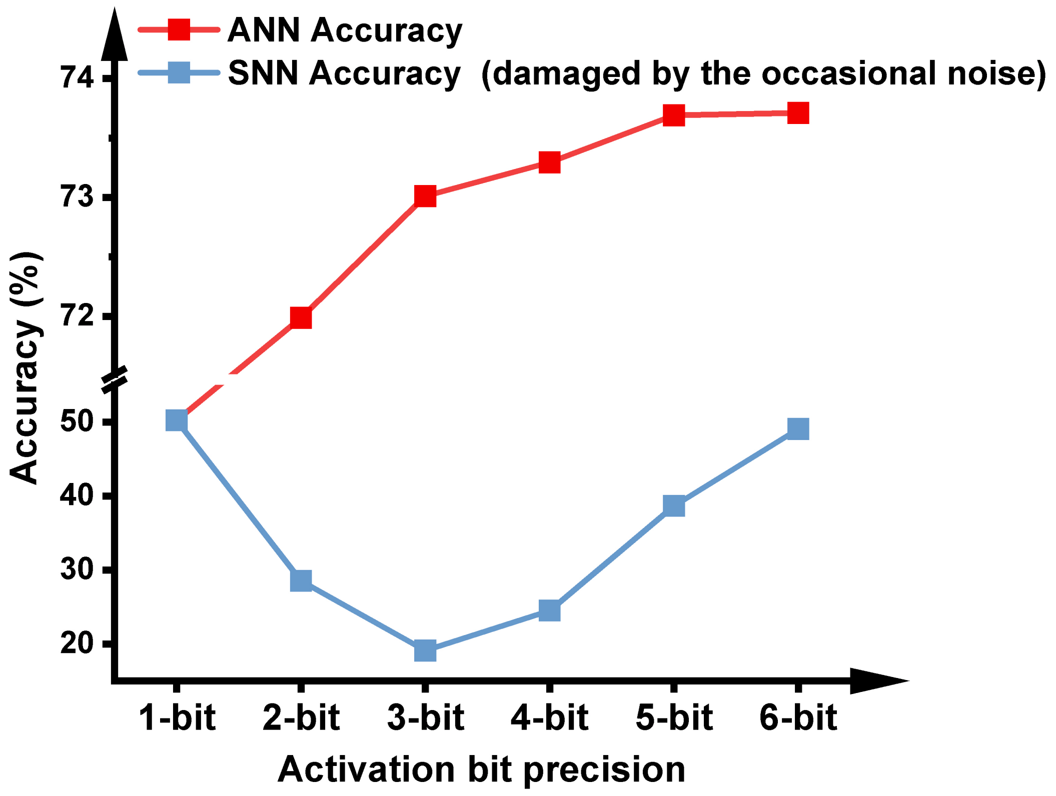

Figure 5.3: ANN accuracy after quantization training with different bit precisions, and SNN accuracy without handling the occasional noise. The model is VGG-16 and the dataset is ImageNet. The ANN-to-SNN conversion is based on the approach proposed in this paper to facilitate the conversion of low-bit activation.

activation quantization precision, and evaluated after converting to SNNs. As shown in Figure 5.3, the SNN accuracy is far from the baseline ANN accuracy for all activation precisions higher than 1 bit. Note that a 1-bit SNN can complete inference in one time step, then its function is equivalent to a 1-bit ANN so its conversion accuracy loss is zero. However, no temporal information was utilized during its inference, so the network is no longer spiking and is out of the scope of fast SNNs. The following section will discuss how this noise may be suppressed to achieve lossless Quant-ANN-to-SNN conversion.

5.4.3 Handling occasional noise and the other three noise types

To achieve lossless Quant-ANN-to-SNN conversion, occasional noise and the other three types of noise need to be handled. This section will briefly introduce how to cope

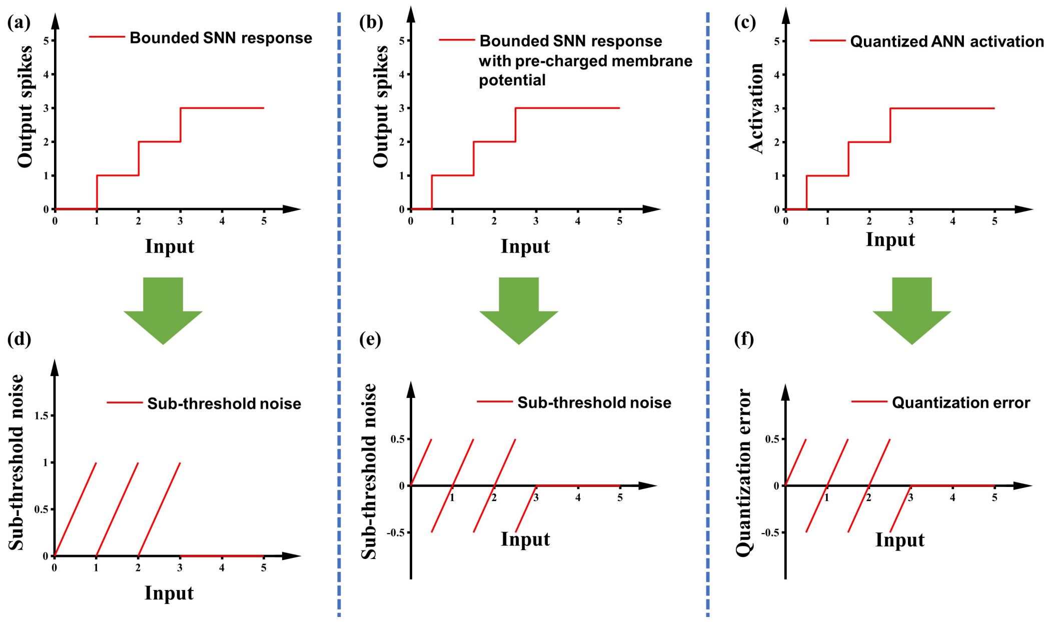

Figure 5.4: (a) The response curve of the modified IF model with the maximum spike count bound of 3; (b) adding pre-charged membrane potential to (a); (c) the response curve of a 2-bit quantified ANN; (d) and (e) are the sub-threshold noise of (a) and (b) respectively, compared with clipped ReLU; (f) is the quantization error of (c) relative to clipped ReLU.

with the other three types of noise, as these three noise types have been researched for years. The focus of this section, then, is illustrating the proposed approach to handle occasional noise.

Handling the first three noise types

By using analog inputs and the modified IF neuron, rate-coding noise is eliminated, and the dropped supra-threshold signal is recovered by the reset-by-subtraction mechanism in the modified IF model [RLH+17].

Sub-threshold noise is the residual membrane potential of spiking neurons after simulation which may cause the output of spiking neurons to be lower than the expected value [DNB+15]. The solutions proposed in previous research are either bias shift [DG21] or pre-charged membrane potential [HCO+21]. This study chooses the pre-charged membrane potential to reduce the amplitude of this noise. After applying the pre-charged membrane potential, the sub-threshold noise shares the same amplitude and patterns as the quantization error in the ANN, so the sub-threshold noise is canceled out. Detailed illustrations are in Figure 5.4.

Algorithm 1 The spiking neuronal model.

Input: Spiking neuron's input x

Parameter: Spiking neuron's voltage u, Current time-step t, Time window T, Threshold th, Generated spike count Z, Maximum spike count Z_max, Heaviside step function Θ

Output: Generated spike z

```
 1: LET $u_{-1} = 0, z_{-1} = 0, Z_{-1} = 0$
 2: for $t \in [0, T]$ do
 3:     $u_t = u_{t-1} - z_{t-1} th + x$
 4:     if $(u_t \geq th) and (Z_{t-1} < Z\_max)$ then
 5:         $z_t = \Theta(u_t - th)$
 6:     else if $(u_t \leq 0) and (Z_{t-1} > 0)$ then
 7:         $z_t = -\Theta(-u_t)$
 8:     else
 9:         $z_t = 0$
10:     end if
11:     $Z_t = Z_{t-1} + z_t$
12: end for
```

Handling occasional noise

To avoid the negative impact of occasional noise, it is necessary to identify its pattern of occurrence. Occasional noise only occurs when:

- The membrane potential of a spiking neuron after the simulation is negative, but at least one spike has been generated during the simulation. This means that more spikes were generated than there should have been. This corresponds to **Situation 2** in the previous section.

- The membrane potential of a spiking neuron after the simulation is higher than the threshold, which means that the generated spike count is lower than expected. This corresponds to **Situation 1** in the previous section.

The spiking neurons that fit either of these two situations are considered to suffer the impact of occasional noise. The occasional noise is mitigated by the proposed feasible method in this study. The main feature of this method is that it can compensate for the occasional noise during the simulation instead of after the simulation. In other words, the event-based nature of SNNs is maintained (An example of a non-event-based SNN can be found in [LZC$^+$22]).

To handle the first situation, a mechanism for generating negative spikes in spiking neurons is added to compensate for the incorrectly emitted positive spikes: A negative spike will be generated when the membrane potential is smaller than zero and the total spike count generated by this neuron is greater than zero. These two prerequisites correspond to the two features listed in the first situation. After a negative spike is generated, the membrane potential will increase by a value equal to its threshold, which is opposite to the reset mechanism of the positive spike. A maximum spike count is set to mimic the maximum quantization value in activation-quantized ANNs. The pseudo-code of this spiking model is given in Algorithm 1. There are two parameters in this spiking model that need to be defined, the threshold of the spiking neurons th and the Maximum spike count limitation Z_max. How these parameters can be determined is illustrated in Section 5.4.5. The simulation time of the SNN is extended correspondingly to enable these newly-generated spikes to propagate to deep layers.

In order to deal with the second situation, the simulation time is simply extended to allow the spike to emit.

5.4.4 Event-based max pooling

Max pooling is problematic for rate-coded SNNs due to their fundamentally different ways of representing information. In ANNs, information is represented by activation values, while in rate-coded SNNs, information is represented by the number of accumulated spikes over time. In each time step, only a limited amount of information is carried by a spike, so simply conducting max pooling on a spike for each time step will introduce considerable accuracy loss. Using winner-take-all mechanisms to model max pooling is more biologically plausible, yet sometimes the winner may not be the one with the maximum activation value.

The following contents discuss this problem and provide a practical approach to implementing max pooling in SNNs. Basically, max pooling in ANNs is picking the maximum value from a series of values in the previous layer. These values are represented as spike counts if the SNNs are rate-coded, so the output of the max pooling should be the maximum value of these spike counts. Note that spike counts need to be recorded after all spikes are generated, to prevent the miscounting of spikes. For example, when calculating the max pooling in a layer of an SNN, assuming the time window of this SNN is T, and the accumulated spike counts recorded at time T is Z_T, then the max pooling output M_T would be

$$M_T = max\_pooling(Z_T), \tag{5.3}$$

where $max\_pooling$ is the max pooling operation. The limitation of this method is obvious. The max pooling output can only be obtained at time T after all spikes have been generated in the previous layer, which violates the event-based nature of SNNs.

To protect the event-based nature of SNNs, some modifications are added to the method above: for each time step t, the max pooling on spike counts Z_t are recorded as

$$M_t = max\_pooling(Z_t). \tag{5.4}$$

The output of max pooling at time t is defined as

$$z_t = M_t - M_{t-1}. \tag{5.5}$$

Using this approach, a max pooling output spike z_t is generated only when M_t changes, which keeps the event-based nature of SNNs. All generated spikes $\sum_{t=1}^{T} z_t$ during the simulation will be the target max pooling output M_T, which is explained below:

According to Equation 5.5, the accumulated spike output of max pooling $\sum_{t=1}^{T} z_t$ would be

$$\begin{aligned}
\sum_{t=1}^{T} z_t &= \sum_{t=1}^{T} (M_t - M_{t-1}) \\
&= (M_T - M_{T-1}) + (M_{T-1} - M_{T-2}) \cdots (M_0 - M_{-1}) \\
&= M_T + (-M_{T-1} + M_{T-1}) + (-M_{T-2} + M_{T-2}) \cdots (-M_0 + M_0) - M_{-1} \\
&= M_T - M_{-1}.
\end{aligned}$$

It can bee seen that all intermediate terms are canceled out, and the last term M_{-1} is 0, this equation then becomes

$$\sum_{t=1}^{T} z_t = M_T. \tag{5.6}$$

The result equals that calculated in Equation 5.3 which conducts max pooling on total spike counts.

Access to the spike count and the calculation of the difference are uncomplicated and can be implemented in PyNN [DBE+09] and PyTorch-based SNN simulation platforms such as snnTorch and SpikingJelly, which offers compatibility with the proposed method. Also, the nature of event-based computing in SNNs is preserved in this proposed method.

Algorithm 2 Event-based max pooling.

Input: The accumulated spike counts of spiking neurons before the max pooling layer Z

Parameter: Max pooling $max\_pooling$, the output of max pooling M, Current time-step t, Time window T

Output: Generated spike z

```
1: LET $z_{-1} = 0, M_{-1} = 0$
2: for $t \in [0, T]$ do
3:     $M_t = max\_pooling(Z_t)$
4:     $z_t = M_t - M_{t-1}$
5: end for
```

5.4.5 Quantization meets ANN-to-SNN conversion

This section provides detailed equations relating to the general ANN-to-SNN conversion and the proposed Quant-ANN-to-SNN conversion. At the end of this section shows that Quant-ANN-to-SNN conversion is a special form of the general ANN-to-SNN conversion.

ANN-to-SNN conversion

In an ANN, the information processing in the artificial neurons in layer l can be modeled as

$$\boldsymbol{y}^l = a(\boldsymbol{W}^l \boldsymbol{y}^{l-1} + \boldsymbol{B}^l), \tag{5.7}$$

where $a(\cdot)$ is the ReLU activation function, $\boldsymbol{W}^l$ and $\boldsymbol{B}^l$ denote the weight and the bias in layer l, $\boldsymbol{y}^l$ is the output of layer l, and $\boldsymbol{y}^{l-1}$ is the output of layer $l-1$ (which is also the input to layer l).

Meanwhile, the integrate-and-fire model used in an SNN is defined as

$$\boldsymbol{u}_t^l = \boldsymbol{u}_{t-1}^l + \widetilde{\boldsymbol{W}^l} \boldsymbol{z}_t^{l-1} \boldsymbol{th}^{l-1} + \widetilde{\boldsymbol{B}^l} - \boldsymbol{z}_{t-1}^l \boldsymbol{th}^l, \tag{5.8}$$

$$\boldsymbol{z}_t^l = \Theta(\boldsymbol{u}_t^l - \boldsymbol{th}^l), \tag{5.9}$$

where $\boldsymbol{u}_t^l$ and $\boldsymbol{u}_{t-1}^l$ are the membrane potential of spiking neurons in layer l at time t and $t-1$ respectively, $\widetilde{\boldsymbol{W}^l}$ is the weight and $\widetilde{\boldsymbol{B}^l}$ is the bias. Θ denotes the Heaviside step function. $\boldsymbol{th}^l$ is the threshold in layer l. $\boldsymbol{z}_t^l$ is the output spike in this layer at time t. Note that the reset mechanism in this spiking neuronal model is the reset-by-subtraction rather than the reset-to-zero.

When conducting ANN-to-SNN conversion based on the data-based normalization, the SNN parameters $\widetilde{\boldsymbol{W}^l}$, $\widetilde{\boldsymbol{B}^l}$ and $\boldsymbol{th}^l$ are calculated by

$$\widetilde{\boldsymbol{W}^l} = \frac{\lambda^{l-1}\boldsymbol{W}^l}{\lambda^l}, \tag{5.10}$$

$$\widetilde{\boldsymbol{B}^l} = \frac{\boldsymbol{B}^l}{\lambda^l}, \tag{5.11}$$

$$\boldsymbol{th}^l = 1, \tag{5.12}$$

where λ^l and λ^{l-1} are the maximum ANN activation value in layer l and the previous layer $l-1$.

Quant-ANN-to-SNN conversion

In an activation-quantized ANN, the activation function is defined by

$$\boldsymbol{y}^l = s^l \times \lfloor clip(\frac{\boldsymbol{W}^l \cdot \boldsymbol{y}^{l-1} + \boldsymbol{B}^l}{s^l}, 0, 2^b - 1) \rceil, \tag{5.13}$$

where s^l is the quantization step size in layer l, and it is the only parameter that is not predefined but is learned during quantization training. b is the activation bit precision so $2^b - 1$ is the maximum quantization value in this ANN. $clip(a, b, c)$ clips a with the value below b set to b and the value above c set to c. $\lfloor a \rceil$ rounds a to the nearest integer. This process comprising scaling, clipping, rounding, and re-scaling is applying a fake quantization to ANN activation.

After conducting Quant-ANN-to-SNN conversion, the applied spiking neuronal model is described in Algorithm 1, or defined by the equations below:

$$\boldsymbol{u}_t^l = \boldsymbol{u}_{t-1}^l + \widetilde{\boldsymbol{W}^l}\boldsymbol{z}_t^{l-1}\boldsymbol{th}^{l-1} + \widetilde{\boldsymbol{B}^l} - \boldsymbol{z}_{t-1}^l\boldsymbol{th}^l, \tag{5.14}$$

$$\boldsymbol{z}_t^l = \Theta(\boldsymbol{u}_t^l - \boldsymbol{th}^l)\Theta(Z\_max - \boldsymbol{Z}_{t-1}^l) - \Theta(-\boldsymbol{u}_t^l)\Theta(\boldsymbol{Z}_{t-1}^l), \tag{5.15}$$

$$\boldsymbol{Z}_t^l = \boldsymbol{Z}_{t-1}^l + \boldsymbol{z}_t^l. \tag{5.16}$$

The meaning of these items is in Algorithm 1. $\Theta(\boldsymbol{u}_t^l - \boldsymbol{th}^l)\Theta(Z\_max - \boldsymbol{Z}_{t-1}^l)$ determines whether a spike is generated, where $\Theta(Z\_max - \boldsymbol{Z}_{t-1}^l)$ prevents emitting more spike than Z_max. $-\Theta(-\boldsymbol{u}_t^l)\Theta(\boldsymbol{Z}_{t-1}^l)$ identifies the occasional noise and compensate it by generating a negative spike. The SNN parameters are calculated by

$$\widetilde{\boldsymbol{W}^l} = \boldsymbol{th}^{l-1}\boldsymbol{W}^l, \tag{5.17}$$

$$\widetilde{\boldsymbol{B}^l} = \boldsymbol{B}^l, \tag{5.18}$$

$$\boldsymbol{th}^l = (2^b - 1)s^l, \tag{5.19}$$

$$Z\_max = 2^b - 1. \tag{5.20}$$

By using the spiking neuronal model described in equations 5.14 - 5.16 and normalizing SNN parameters by Equations 5.17 - 5.20, a lossless Quant-ANN-to-SNN conversion can be achieved.

Connection between ANN-to-SNN conversion and Quant-ANN-to-SNN conversion

The connection between ANN-to-SNN conversion and Quant-ANN-to-SNN conversion is illustrated below. Equations 5.17 - 5.19 are significantly different from the data-based normalization Equations 5.10, 5.11 and 5.12. However, one characteristic of spiking neural networks is that the function of an SNN will be unchanged after scaling weights, bias, and spiking thresholds simultaneously. If scaling these parameters by $1/(2^b - 1)s^l$ simultaneously, these equations become

$$\widetilde{\boldsymbol{W}^l} = \frac{(2^b - 1)s^{l-1}\boldsymbol{W}^l}{(2^b - 1)s^l}, \tag{5.21}$$

$$\widetilde{\boldsymbol{B}^l} = \frac{\boldsymbol{B}^l}{(2^b - 1)s^l}, \tag{5.22}$$

$$\boldsymbol{th}^l = 1. \tag{5.23}$$

It is apparent that these equations become more similar to the data-based normalization Equations 5.10, 5.11, and 5.12. For instance, $(2^b - 1)s^l$ in Equation 5.21 corresponds to λ^l in Equation 5.10, and they are both the maximum output value in an ANN layer. This

Table 5.2: Hyper-parameters of ANN quantization training.

Learning rate	0.01
Momentum	0.9
Weight decay	0.0005
Epoch	40
Batch size	32
Other technique	Data augmentation

shows the internal correspondence of the described method to traditional ANN-to-SNN conversion techniques.

5.5 Experiments

5.5.1 Experimental setup

The quantization training is applied on pre-trained full precision VGG-16 and ResNet models. The networks were trained by stochastic gradient descent with the loss function of cross-entropy and the exponential decay scheduler. Detailed hyper-parameters are in Table 5.2. A 2-bit activation precision is chosen in all hidden layers for CIFAR-10 and ImageNet in quantization training to render the best SNN latency. Notably, the output layer is 3-bit, and the reasons for this choice are discussed in the following section. Both ANN quantization training and SNN implementation are carried out with PyTorch.

In SNN simulation, the network input is analog-coded [RLH+17] and the time resolution is 1ms. The adopted neuronal model was described in Algorithm 1. The maximum spike count is limited to $2^b - 1$ in hidden layers and $2^{b+1} - 1$ in the output layer, which corresponds to the maximum quantization states during quantization training. b is the activation bit precision during ANN quantization training and is chosen as 2. The weight $\widetilde{\boldsymbol{W}}^l$, the bias $\widetilde{\boldsymbol{B}}^l$, the threshold $\boldsymbol{th}^l$ and the maximum spike count limitation Z_max in layer l are determined by

$$\widetilde{\boldsymbol{W}^1} = \boldsymbol{th}^{l-1}\boldsymbol{W}^l, \tag{5.24}$$

$$\widetilde{\boldsymbol{B}^1} = \boldsymbol{B}^l, \tag{5.25}$$

Table 5.3: Benchmarking SNNs built by ANN-to-SNN conversion on CIFAR-10 and ImageNet.

Method	**Dataset**	**Architecture**	**Acc(ANN)(%)**	**Acc(SNN)(%)**	**Latency (ms)**
RNL+RIL [DYTH21]	CIFAR-10	ResNet-18	93.06	91.96	64
RNL+RIL [DYTH21]	CIFAR-10	VGG-16	92.82	91.15	64
TCL [HC20]	CIFAR-10	ResNet-20	91.58	91.22	35
TCL [HC20]	CIFAR-10	VGG-16	93.25	92.6	20
QCFS [BFD+21]	CIFAR-10	ResNet-20	91.77	91.62	16
QCFS [BFD+21]	CIFAR-10	ResNet-18	**96.04**	94.82	8
QCFS [BFD+21]	CIFAR-10	VGG-16	95.52	**94.95**	8
TS [DG21]	CIFAR-10	ResNet-20	92.32	92.41	16
TS [DG21]	CIFAR-10	VGG-16	92.09	92.29	16
QFFS (This work)	CIFAR-10	ResNet-18	93.12	93.14	**4**
QFFS (This work)	CIFAR-10	VGG-16	92.44	92.64	**4**
Spike-Norm [SYW+19]	ImageNet	VGG-16	70.52	69.96	2500
Spike-Norm [SYW+19]	ImageNet	ResNet-34	70.69	65.47	2000
RMP [HSR20]	ImageNet	VGG-16	73.49	73.09	4096
RMP [HSR20]	ImageNet	ResNet-34	70.64	69.89	4096
TCL [HC20]	ImageNet	VGG-16	73.22	70.75	30
TCL [HC20]	ImageNet	ResNet-34	70.85	70.37	250
QCFS [BFD+21]	ImageNet	VGG-16	74.29	72.85	64
QCFS [BFD+21]	ImageNet	ResNet-34	**74.32**	72.35	64
TS [DG21]	ImageNet	VGG-16	72.4	70.97	64
QFFS (This work)	ImageNet	ResNet-50	70.15	70.18	8
QFFS (This work)	ImageNet	VGG-16	69.88	69.69	8
QFFS with analog output (This work)	ImageNet	ResNet-50	72.81(74.07)	72.91(**74.36**)	5(10)
QFFS with analog output(This work)	ImageNet	VGG-16	71.88(73.08)	72.10(73.10)	**4**(8)

$$\boldsymbol{th}^l = (2^b - 1)s^l, \tag{5.26}$$

$$Z\_max = 2^b - 1, \tag{5.27}$$

where $\boldsymbol{W}^l$, $\boldsymbol{B}^l$ and s^l are the weight, the bias and the quantization step size in layer l which are learned during ANN quantization training [EMB+19]. The membrane potential of the spiking neurons in all layers is pre-charged by $0.5\boldsymbol{th}$ at the first time step to eliminate systematic errors relative to the quantized ANNs [HCO+21].

Bias in the SNNs is modeled by constant current injection into the spiking neurons [RLH+17]. After time step $2^b - 1$, both the bias and the network input are shut down, so only the spikes caused by occasional noise can be passed through the network.

5.5.2 Benchmark results

The following benchmark results are on ResNet models. More results on VGG-16 models are in Section 5.5.6.

After converting quantized ANNs to SNNs, an accuracy of 93.14% is achieved

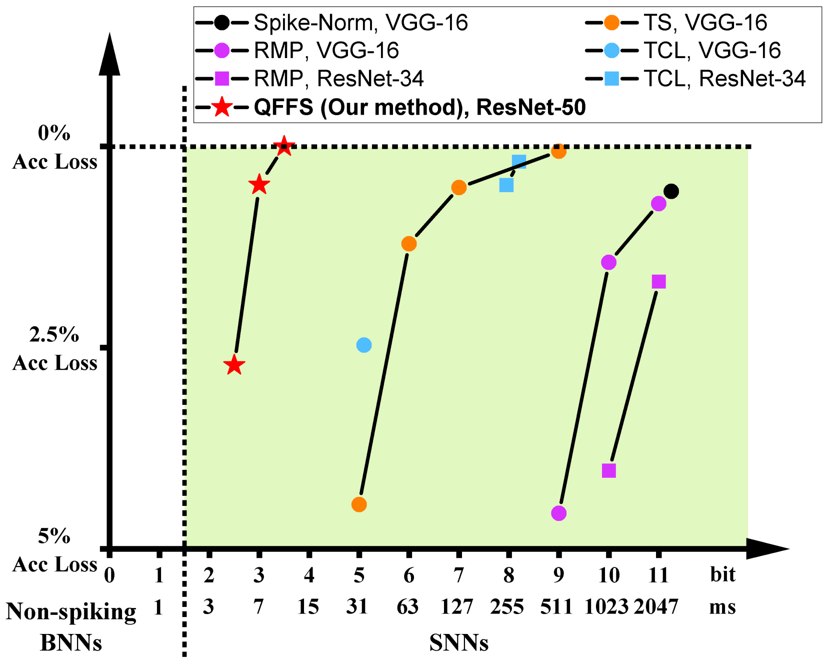

Figure 5.5: The relationship between ANN-to-SNN conversion loss and SNN inference latency on ImageNet.

within 4 time steps on CIFAR-10 and an accuracy of 70.18% is reached within 8 time steps on ImageNet. Compared with previous work on ANN-to-SNN conversion, the inference latency of the SNNs is shortened significantly while retaining competitive accuracy as shown in Table 5.3.

Note that in fast SNN research, sometimes the SNN accuracy will be higher than the ANN accuracy. This phenomenon usually appears when an SNN is converted from an ANN whose activation function is clipped ReLU, or when the dataset is less challenging such as CIFAR-10. This has been reported in several studies but an adequate explanation is still lacking [DG21, DYTH21]. This research also shows higher SNN accuracy than that in the ANN in some experimental settings - the extreme case is that the accuracy of a SNN is about 0.3% higher than its ANN accuracy, as shown in Table 5.3. The reason may be that the applied ANN activation function contains a clipped point such as in the clipped ReLU. Another potential reason may be the difference in information processing mechanisms between ANNs and SNNs. ANNs calculate the activation function by multiplication, while SNNs calculate outputs using their integrate-and-fire mechanism. Thus, even if a perfect ANN-to-SNN conversion is conducted, a value in the ANN may be represented as a slightly different value in the SNN.

This study also benchmarks the required time steps to achieve lossless ANN-to-SNN

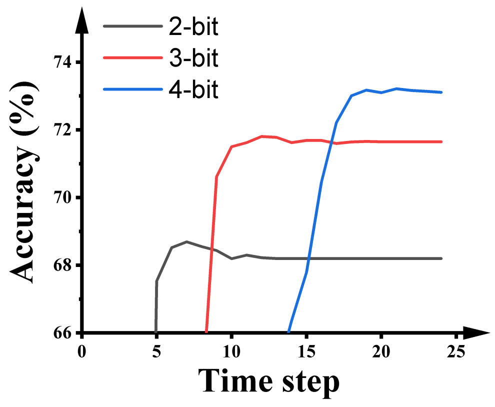

Figure 5.6: The impact of the activation precision during the quantization training of ResNet-50 on the performance of SNNs on ImageNet.

conversion on ImageNet in Figure 5.5. The y-axis in this figure represents the accuracy gap to the baseline ANN accuracy before the conversion. The horizontal axis is the required number of time steps of the SNNs, which is converted into the equivalent bit resolution at the bottom. What stands out in this figure is that the proposed quantization framework for fast SNNs only needs 13 time steps - about 4 bits of information - to reach lossless accuracy, while other methods need at least 500 time steps, or 9 bits of information, to achieve lossless accuracy. This highlights the merit of applying information compression techniques to SNNs and the effectiveness of the proposed Quant-ANN-to-SNN conversion paradigm.

5.5.3 Bit precision during quantization training

Figure 5.6 illustrates the impact of the activation precision during quantization training on the accuracy and latency of SNNs on ImageNet. As shown in the figure, higher activation precision during quantization training will offer higher accuracy in the SNNs, while the number of time steps required by the SNNs is extended. Hence, there is an accuracy-latency trade-off inside the Quant-ANN-to-SNN conversion technique.

Some SNN research keeps the output layer as floating-point and suggests that this promotes inference accuracy. For a fair comparison with this kind of research, the performance of SNNs built using this paradigm is also reported. As shown in Figure 5.7, using full precision in the output layer during ANN quantization training obviously improves the inference accuracy and latency of the SNNs. Particularly, the 2-bit SNN reaches 72.91% in 5 time steps, and the 3-bit SNN reaches 74.36% in 10 time steps.

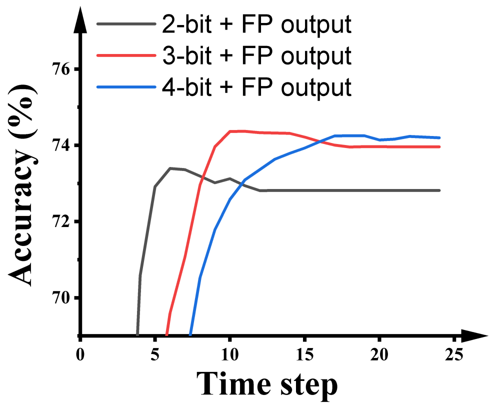

Figure 5.7: The impact of the activation precision in hidden layers during the quantization training of ResNet-50 on the performance of SNNs on ImageNet.

5.5.4 Bit precision in the output layer

The results in the previous section show that the network accuracy is very sensitive to the bit precision in the output layer. For instance, changing the output layer from 2-bit to floating-point gives a 4% accuracy improvement on ImageNet. This section provides more fine-grained results on the bit precision in the output layer. Also, how to improve inference accuracy without sacrificing biological plausibility is discussed, in particular without using analog neurons in the output layer of the SNNs.

From Figure 5.6, it can be seen that the SNN converted from a 2-bit ANN needs about 7 time steps to reach the highest accuracy, which means that most of the information has been transmitted to the output layer in 7 time steps. 7 time steps can represent 3 bits of information from a rate-coded spiking neuron in theory, while the bit precision of the output layer is only 2 bits. This gap motivates us to further utilize the information representation ability of spiking neurons by adjusting the precision of the output layer.

The bit precision in all hidden layers is kept as 2-bit, and the bit precision in the output layer is adjusted before quantization training. As shown in Figure 5.8, higher precision in the output layer brings higher accuracy but longer latency. What stands out in this figure is the case with 3-bit bit precision: its accuracy improves 1.49% while its latency is only extended by 1 time step compared with the case with 2-bit precision. Meanwhile, its latency is 2 times shorter than the case with 4-bit precision. With that in mind, the presented results choose 2-bit in all hidden layers and 3-bit in the output layer during quantization training to build high-accuracy, low-latency SNNs.

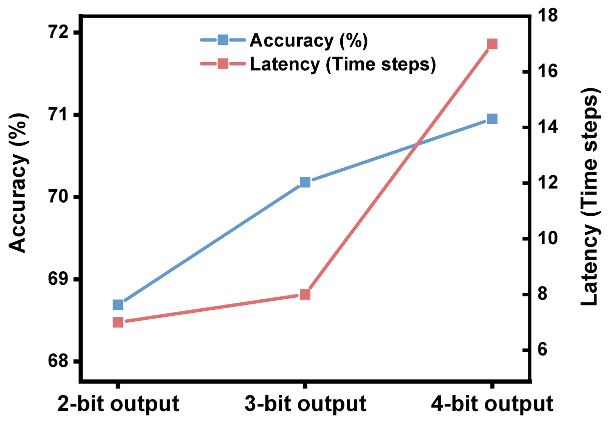

Figure 5.8: The impact of the activation precision in the output layer during ANN quantization on the performance of SNNs on ImageNet. The network architecture is ResNet-50.

5.5.5 Ablation studies

The effect of the proposed methods are decomposed using an ablation study as shown in Table 5.4. The default setting is the SNN converted from a 2-bit quantized ANN. After applying the mechanisms of generating negative spikes and extending the simulation time to let the newly generated spikes propagate, the accuracy reaches 67.29%. Using max pooling in the network increases the SNN accuracy to 68.69% compared with using average pooling. Another 1.49% accuracy improvement comes from using 3-bit quantization in the output layer.

5.5.6 Results on VGG-16

Table 5.4: Ablation studies on ImageNet

Index	**Setting**	Accuracy
①	**Default**	0.01%
②	① **+ Negative spike**	10.64%
③	② **+ Simulation time step extension**	67.29%
④	③ **+ Max pooling**	68.69%
⑤	④ **+ More bits in the output layer**	70.18%

The SNNs applying the VGG-16 network architecture achieved an accuracy of 92.62% in 4 time steps on CIFAR-10 and an accuracy of 69.69% in 8 time steps on ImageNet. The results on the impact of bit precision, the bit precision in the hidden layers, and the bit precision in the output layer are shown in Figure 5.9.

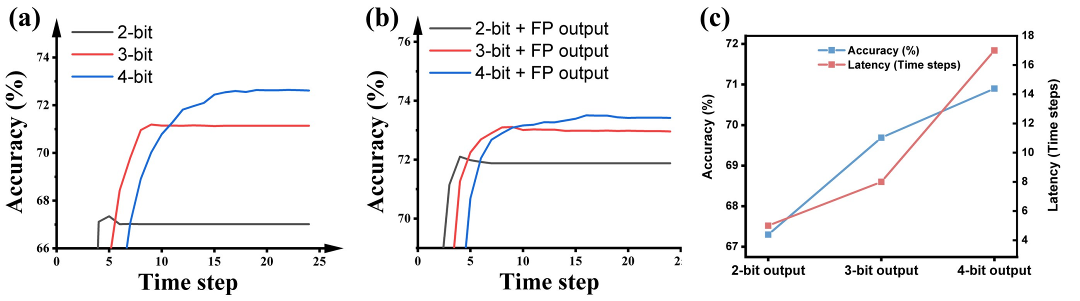

Figure 5.9: The impact of the bit precision in (a) all layers, (b) all hidden layers, and (c) the output layer on SNN accuracy and latency. The output precision in (b) is floating-point; the precision of hidden layers in (c) is 2 bits. The dataset is ImageNet and the network architecture is VGG-16.

5.6 Comparison with other fast SNN approaches

This section summarizes and compares three methods to build a low-latency SNN.

Quant-ANN-to-SNN conversion: This method is based on the observation that after ANN-to-SNN conversion, the ANN activation precision is represented as SNN temporal precision. Therefore, a low-latency SNN can be achieved by converting a low activation precision ANN. The fundamental challenges of this method are low-bit quantization training in ANNs and noise suppression in SNNs. The first challenge can be solved by applying advanced ANN quantization techniques. The second challenge, currently, has not got an efficient solution. The current approaches either ignore the occasional noise in SNNs so meet a huge accuracy drop [BFD+21], or adopt negative spikes to correct the occasional noise so bring extra computational overhead.

Tandem learning [WXH+21] (SNN calibration [LDD+21]): This method reduces SNN latency by minimizing the differences between ANN activation and SNN output and regarding it as a optimization problem that can be solved by gradient descent. The first step of tandem learning is adopting a pre-trained ANN which will not bring extra overhead. However, the next step of tandem learning, layer-wise fine-tuning, needs a large number of epochs, and the number of epochs is related to the network depth. A loss function is defined during fine-tuning to model the difference between the activation value of analog neurons and the output spike counts of the corresponding spiking neurons; a gradient descent algorithm is then applied layer-by-layer to optimize this loss function.

Direct training by surrogate gradients: This method applies a technique in recurrent neural networks called back propagation through time to SNNs, which enables training SNNs directly by gradient descent. In direct training, a SNN is unfolded at the time scale and a surrogate gradient is adopted to alleviate the non-linearity problem of spike generating. In this way, the gradients can backpropagate in SNNs and SNNs can be trainable. Direct training has been studied by many researcher, and many new techniques has been proposed to boost the performance of direct training [NMZ19, WDL$^+$18, FYC$^+$21a].

Quant-ANN-to-SNN conversion and Tandem learning are all based on ANN-to-SNN conversion techniques. Both of these two methods reduce SNN latency by improving the similarity of artificial neurons' activation and spiking neuron's output spike counts. Specifically, Quant-ANN-to-SNN conversion improves the ANN-SNN similarity by quantizing ANNs, while Tandem learning improves the ANN-SNN similarity by fine-tuning SNNs.

Both tandem learning and direct training conduct training according to the simulation results of a SNN. However, Tandem learning only records output spike counts, but direct training requires recording more parameters during feedforward propagation and backpropagation. Tandem learning trains SNNs layer-wise, while direct training does not.

To further clarify the aforementioned three methods of reducing SNN latency. These methods are compared in the aspect of resource budget during training, and performance during inference.

Resource budget during training: The resource budget of Quant-ANN-to-SNN conversion during training is very low, as it just needs a short ANN quantization training (e.g. only 30 epochs) based on a full-precision ANN model. Tandem learning needs to simulate an SNN layer-wise during training. When the network is deep, it will require huge memory and computational resource. Direct training is the most computationally expensive one among these three methods, it requires more fine-grained parameter recording of an SNN (e.g. membrane potential and spike generating in each spiking neurons and each time step) during training, and one extra temporal dimension to consider during training.

Performance during inference: Currently, the highest accuracy and latency are achieved by Quant-ANN-to-SNN conversion. For example, it can achieve an accuracy of 72.1% in 4 time steps on ImageNet (with the compromise of adopting a less biologically plausible spiking neuronal model). Direct training is capable of achieving a similar latency but lower accuracy than Quant-ANN-to-SNN conversion. There are not reported results to achieve similar latency and accuracy by tandem learning.

5.7 Further improvements

This research provides a novel method to achieve lossless ANN-to-SNN conversion within several time steps. Furthermore, a framework to facilitate future improvements in fast SNN research is built: The accuracy can be increased by improving the first step in QFFS, which is quantizing the ANN activation as shown in Figure 5.2. The method chosen for quantization training is LSQ. However, quantization training techniques are developing rapidly, and there are other effective methods being proposed after LSQ. This study suggest that the progress in ANN quantization techniques can promote accuracy improvements in fast SNN research through the bridge built by the proposed framework.

The four identified types of noise are the main cause of the degradation in accuracy and the extension in latency. In this research, the noise suppression measures are considered only after the ANN-to-SNN conversion. An alternative method is to suppress noise before conducting the ANN-to-SNN conversion. For example, it may be helpful if some constraints can be added during quantization training to make SNNs robust to occasional noise. In this case, the noise could be suppressed more effectively, and the latency of SNNs may be improved.

Furthermore, occasional noise is suppressed by modifying spiking neuronal models (the third step in Figure 5.2). It is worthwhile to study how to use more biologically plausible mechanisms to suppress occasional noise in the future, thereby further improving the proposed QFFS.

5.8 Summary

By establishing a bridge from ANN quantization precision to SNN inference latency, this study achieved state-of-the-art inference latency in SNNs. This demonstration significantly improves the performance of rate-coded SNNs, and should facilitate future SNN implementations on edge devices for ultra-fast, event-based computing. SNNs encoded by temporal coding may also benefit from this research as, in these encoding schemes, the amount of information to be encoded is crucial as well.

This study offers a fresh perspective on how to generate more rapid progress on SNNs: in the proposed quantization framework for fast SNNs, the first step is selecting one effective technique (instead of developing an SNN algorithm from scratch). The remaining three steps in QFFS are making the knowledge transmission from ANNs to SNNs smoother. Considering the prosperity in current ANN research, it is highly possible that this research concept will continue to work in the near future.

Chapter 6

Summary and future work

6.1 Summary

Central to this thesis is the optimization of rate-based spiking neural networks, and it is related to three different but overlapped research fields: deep learning, neuromorphic hardware, and computational neuroscience.

Chapter 5 investigates the quantization techniques in deep learning, and applies them in SNNs to minimize latency. This chapter proposes a methodology for optimizing low-latency SNNs, and suggests the necessity of knowledge transmission from deep learning to SNNs. Chapter 3 investigates noise, an element that has been intensively researched in neuroscience and analog neuromorphic hardware, in the context of algorithm optimization in SNNs, which provides a useful metric to SNN algorithms. Also, Chapter 3 demonstrates the deployment of an SNN algorithm to neuromorphic hardware and the hardware-algorithm co-design for SNNs with noisy synapses. The main issue addressed in Chapter 4 is retaining biological plausibility in the current SNN algorithms. By maintaining the biological plausibility, it facilitates SNNs to get inspiration from the observations in computational neuroscience. Chapter 2 is concerned with the basic background in this thesis.

The SNNs researched in this thesis are trained by ANN-to-SNN conversion which converts the current non-spiking neural network models to spiking neural networks. ANN-to-SNN conversion has proven to be a highly-efficient method to rapidly build a functional SNN, so various optimizations on SNNs can be applied and analyzed. Though ANN-to-SNN conversion and rate coding are dominant methods in SNNs, there are also other training methods and encoding strategies in SNNs as introduced in Chapter 2. However, this thesis focuses on rate-coded SNNs which are trained by

ANN-to-SNN conversion, since these two methods have enabled research on SNN optimizations. The insights generated in the studies presented in this thesis can benefit other methods. For instance, the concept of information compression analyzed in Quant-ANN-to-SNN conversion can benefit temporal coding; also, the performance achieved in ANN-to-SNN conversion and provide a baseline to SNNs trained by surrogate gradients.

6.2 Future work

Despite the promising results presented in this thesis, there are still some research questions that could be asked.

Further research should be undertaken to further investigate the noisy synaptic weights in SNNs. Firstly, the investigated neural network structures are shallow FCNs and CNNs, and the benchmark used in this research is a basic computer vision benchmark MNIST. More experiments in deeper neural network architectures [SZ14] and harder benchmarks [KH^{+}09, RDS^{+}15] are required. Secondly, in this study, only the robustness of Gaussian noise is investigated in neural networks, and the standard deviation of Gaussian noise is proportional to the value of the synaptic weights. In practice, the random noise found in advanced materials may have different distributions. Future work is required to establish systematic studies of different noise types. Also, to conduct more systematic studies, a potential research direction is expanding results to different spiking neuron models (e.g. leaky integrate-and-fire model) and different encoding methods (e.g. temporal coding). Thirdly, the inference accuracy and the inference latency are different for FCNs and CNNs for the same noise level. According to the results in previous sections, the accuracy of CNNs drops faster than FCNs with the increase of noise level for both ANNs and SNNs. Meanwhile, the inference latency of SNNs grows faster in CNNs than in FCNs. These different performances show the different tolerances to noisy weights in FCNs and CNNs and stem from their different neural network architectures, which have not been thoroughly explored in this study.

In the research on the skyrmionic synapses, an application scenario is proposed to detach training and inference for edge intelligence tasks. More detailed illustrations on this topic are necessary. The research into SNN implementation on SpiNNaker achieved state-of-the-art accuracy and conversion loss on MNIST on SpiNNaker. Further improvements in resource management and latency optimizations will need to be undertaken.

A new firing rate normalization strategy referred to as MCR-Norm is proposed in

Chapter 4 to balance performance and biological plausibility in SNNs. The firing rate is normalized to 400 Hz according to the quantitative experiments. Future work is required to allocate gradients on the firing rate range, by which SNNs can learn their optimal spike rates during ANN-to-SNN conversion.

Chapter 5 provides a full picture of establishing a quantization framework for fast SNNs. The detailed model applied in Quant-ANN-to-SNN conversion is a modified IF model. It is worth exploring achieving state-of-the-art latency by the standard IF model and rate coding.

Bibliography

[Abb99] Larry F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999.

[ANDL18] Jithendar Anumula, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. Feature representations for neuromorphic audio spike streams. *Frontiers in neuroscience*, 12:23, 2018.

[ATB$^+$17] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.

[AZ26] Edgar D Adrian and Yngve Zotterman. The impulses produced by sensory nerve-endings: Part ii. the response of a single end-organ. *The Journal of physiology*, 61(2):151, 1926.

[BBH$^+$14] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48, 2014.

[BBNM11] Lars Buesing, Johannes Bill, Bernhard Nessler, and Wolfgang Maass. Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS computational biology*, 7(11):e1002211, 2011.

[BCC$^+$15] Michael Beyeler, Kristofor D Carlson, Ting-Shuo Chou, Nikil Dutt, and Jeffrey L Krichmar. Carlsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[BCFA01] Nicolas Brunel, Frances S Chance, Nicolas Fourcaud, and Larry F Abbott. Effects of synaptic noise and filtering on the frequency response of spiking neurons. *Physical Review Letters*, 86(10):2186, 2001.

[BCHE19] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th annual neuro-inspired computational elements workshop*, pages 1–8, 2019.

[BFD$^+$21] Tong Bu, Wei Fang, Jianhao Ding, PengLin Dai, Zhaofei Yu, and Tiejun Huang. Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International Conference on Learning Representations*, 2021.

[BSS$^+$20] 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.

[CATVS17] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[CCK15] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

[CGR21] Sayeed Shafayet Chowdhury, Isha Garg, and Kaushik Roy. Spatio-temporal pruning and quantization for low-latency spiking neural networks. *arXiv preprint arXiv:2104.12528*, 2021.

[CKL$^+$21] Jonathan Cornford, Damjan Kalajdzievski, Marco Leite, Amélie Lamarquette, Dimitri M Kullmann, and Blake Richards. Learning to live with dale's principle: Anns with separate excitatory and inhibitory units. *bioRxiv*, pages 2020–11, 2021.

[CLL$^+$20] Runze Chen, Chen Li, Yu Li, James J Miles, Giacomo Indiveri, Steve Furber, Vasilis F Pavlidis, and Christoforos Moutafis. Nanoscale room-temperature multilayer skyrmionic synapse for deep spiking neural networks. *Physical Review Applied*, 14(1):014096, 2020.

[CSR18] Mei-Chin Chen, Abhronil Sengupta, and Kaushik Roy. Magnetic skyrmion as a spintronic deep learning spiking neuron processor. *IEEE Transactions on Magnetics*, 54(8):1–7, 2018.

[CSSZ20] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[DBE$^+$09] Andrew P Davison, Daniel Brüderle, Jochen M Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, 2:11, 2009.

[DC15] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DG21] Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. *arXiv preprint arXiv:2103.00476*, 2021.

[DNB$^+$15] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. ieee, 2015.

[DSL+18] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[DYTH21] Jianhao Ding, Zhaofei Yu, Yonghong Tian, and Tiejun Huang. Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. *arXiv preprint arXiv:2105.11654*, 2021.

[EAM+15] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in neural information processing systems*, pages 1117–1125, 2015.

[EEVG+15] Mark Everingham, SM Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[EMB+19] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[ESC+12] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

[EWN+21] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

[FCD+20] Wei Fang, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Yonghong Tian, and other contributors. Spikingjelly. `https://github.com/fangwei123456/spikingjelly`, 2020. Accessed: YYYY-MM-DD.

[FFFP04] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pages 178–178. IEEE, 2004.

[FGTP14] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[FLP$^+$12] Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE transactions on computers*, 62(12):2454–2467, 2012.

[FSW08] A Aldo Faisal, Luc PJ Selen, and Daniel M Wolpert. Noise in the nervous system. *Nature reviews neuroscience*, 9(4):292–303, 2008.

[FYC$^+$21a] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *arXiv preprint arXiv:2102.04159*, 2021.

[FYC$^+$21b] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2661–2671, 2021.

[GB08] Dan FM Goodman and Romain Brette. Brian: a simulator for spiking neural networks in python. *Frontiers in neuroinformatics*, 2:5, 2008.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[GD07] Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[GDO$^+$20] 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.

[GFES21] Wenzhe Guo, Mohammed E Fouda, Ahmed M Eltawil, and Khaled Nabil Salama. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15:638474, 2021.

[GJDSA07] Elena Garcia, Maria Antonia Jimenez, Pablo Gonzalez De Santos, and Manuel Armada. The evolution of robotics research. *IEEE Robotics & Automation Magazine*, 14(1):90–103, 2007.

[GKNP14] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[HC20] Nguyen-Dong Ho and Ik-Joon Chang. Tcl: an ann-to-snn conversion with trainable clipping layers. *arXiv preprint arXiv:2008.04509*, 2020.

[HCO$^+$21] Sungmin Hwang, Jeesoo Chang, Min-Hye Oh, Kyung Kyu Min, Taejin Jang, Kyungchul Park, Junsu Yu, Jong-Ho Lee, and Byung-Gook Park. Low-latency spiking neural networks using pre-charged membrane potential and delayed evaluation. *Frontiers in Neuroscience*, 15:135, 2021.

[HE15] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

[HH52] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[HRFS16] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037*, 2016.

[HSK$^+$18] Hananel Hazan, Daniel J Saunders, Hassaan Khan, Devdhar Patel, Darpan T Sanghavi, Hava T Siegelmann, and Robert Kozma. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in neuroinformatics*, 12:89, 2018.

[HSR20] Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13558–13567, 2020.

[HZRS15] 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.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IC20] Nabil Imam and Thomas A Cleland. Rapid online learning and robust recall in a neuromorphic olfactory circuit. *Nature Machine Intelligence*, 2(3):181–191, 2020.

[Izh03] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[Izh04] Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

[JLPPSRE22] Alejandro Juarez-Lora, Victor H Ponce-Ponce, Humberto Sossa, and Elsa Rubio-Espino. R-stdp spiking neural network architecture for motion control on a changing friction joint robotic arm. *Frontiers in Neurorobotics*, 16, 2022.

[JUZ$^+$15] Wanjun Jiang, Pramey Upadhyaya, Wei Zhang, Guoqiang Yu, M Benjamin Jungfleisch, Frank Y Fradin, John E Pearson, Yaroslav Tserkovnyak, Kang L Wang, Olle Heinonen, et al. Blowing magnetic skyrmion bubbles. *Science*, 349(6245):283–286, 2015.

[KGP22] Laura Kriener, Julian Göltz, and Mihai A Petrovici. The yin-yang dataset. In *Neuro-Inspired Computational Elements Conference*, pages 107–111, 2022.

[KGT$^+$15] Dion Khodagholy, Jennifer N Gelinas, Thomas Thesen, Werner Doyle, Orrin Devinsky, George G Malliaras, and György Buzsáki. Neurogrid: recording action potentials from the surface of the brain. *Nature neuroscience*, 18(2):310–315, 2015.

[KGTM18] 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.

[KH$^+$09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KMN20] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020.

[KPNY20] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11270–11277, 2020.

[Kri18] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LCF17] Qian Liu, Yunhua Chen, and Steve Furber. Noisy softplus: an activation function that enables snns to be trained as anns. *arXiv preprint arXiv:1706.03609*, 2017.

[LCMF20] Chen Li, Runze Chen, Christoforos Moutafis, and Steve Furber. Robustness to noisy synaptic weights in spiking neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[LCTA16] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):1–10, 2016.

[LD93] R Gary Leonard and George Doddington. Tidigits speech corpus. *Texas Instruments, Inc*, 1993.

[LDD$^+$21] Yuhang Li, Shikuang Deng, Xin Dong, Ruihao Gong, and Shi Gu. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. *arXiv preprint arXiv:2106.06984*, 2021.

[LDP16] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[LDQ$^+$21] Yihan Lin, Wei Ding, Shaohua Qiang, Lei Deng, and Guoqi Li. Es-imagenet: A million event-stream classification dataset for spiking neural networks. *Frontiers in Neuroscience*, page 1546, 2021.

[Len03] Peter Lennie. The cost of cortical computation. *Current biology*, 13(6):493–497, 2003.

[LF16] Qian Liu and Steve Furber. Noisy softplus: A biology inspired activation function. In *International Conference on Neural Information Processing*, pages 405–412. Springer, 2016.

[LF21] Chen Li and Steve Furber. Towards biologically-plausible neuron models and firing rates in high-performance deep spiking neural networks. In *International Conference on Neuromorphic Systems 2021*, pages 1–7, 2021.

[LGZ$^+$21] Yuhang Li, Yufei Guo, Shanghang Zhang, Shikuang Deng, Yongqing Hai, and Shi Gu. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[LLJ$^+$17] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in neuroscience*, 11:309, 2017.

[LN21] Hin Wai Lui and Emre Neftci. Hessian aware quantization of spiking neural networks. *arXiv preprint arXiv:2104.14117*, 2021.

[LPSR18] Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in neuroscience*, 12:435, 2018.

[LvSMD13] Shih-Chii Liu, André van Schaik, Bradley A Minch, and Tobi Delbruck. Asynchronous binaural spatial audition sensor with 4 x 64 x 4 channel output. *IEEE transactions on biomedical circuits and systems*, 8(4):453–464, 2013.

[LWM$^+$18] Yibo Li, Zhongrui Wang, Rivu Midya, Qiangfei Xia, and J Joshua Yang. Review of memristor devices in neuromorphic computing: materials sciences and device challenges. *Journal of Physics D: Applied Physics*, 51(50):503002, 2018.

[LZC$^+$22] Fangxin Liu, Wenbo Zhao, Yongbiao Chen, Zongwu Wang, and Li Jiang. Spikeconverter: An efficient conversion framework zipping the gap between artificial neural networks and spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[LZZ21] Yang Li, Yi Zeng, and Dongcheng Zhao. Bsnn: Towards faster and better conversion of artificial neural networks to spiking neural networks with bistable neurons. *arXiv preprint arXiv:2105.12917*, 2021.

[MAAI$^+$14] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[Mah92] Misha Mahowald. Vlsi analogs of neuronal visual processing: a synthesis of form and function. 1992.

[Mea90] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[MGNDM19] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, and Timothée Masquelier. Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron. *Frontiers in neuroscience*, page 625, 2019.

[MHAK21] Etienne Mueller, Julius Hansjakob, Daniel Auge, and Alois Knoll. Minimizing inference time: Optimization methods for converted deep spiking neural networks. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[MJS07] Wolfgang Maass, Prashant Joshi, and Eduardo D Sontag. Computational aspects of feedback in neural circuits. *PLoS Comput Biol*, 3(1):e165, 2007.

[MPSC17] Hesham Mostafa, Bruno U Pedroni, Sadique Sheik, and Gert Cauwenberghs. Fast classification using sparsely active spiking networks. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[Mur98] Venkatesh N Murthy. Synaptic plasticity: step-wise strengthening. *Current biology*, 8(18):R650–R653, 1998.

[NAPD17] Emre O Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in neuroscience*, 11:324, 2017.

[NBLD22] Simon Narduzzi, Siavash A Bigdeli, Shih-Chii Liu, and L Andrea Dunbar. Optimizing the consumption of spiking neural networks with activity regularization. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 61–65. IEEE, 2022.

[NHA$^+$00] Satoshi Nakamura, Kazuo Hiyane, Futoshi Asano, Takanobu Nishiura, and Takeshi Yamada. Acoustical sound database in real environments for sound scene understanding and hands-free speech recognition. 2000.

[NMZ19] 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.

[NWC$^+$11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[OB11] Srdjan Ostojic and Nicolas Brunel. From spiking neuron models to linear-nonlinear models. *PLoS computational biology*, 7(1):e1001056, 2011.

[OJCT15] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.

[ONL$^+$13] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in neuroscience*, 7:178, 2013.

[OW16] Peter O'Connor and Max Welling. Deep spiking networks. *arXiv preprint arXiv:1602.08323*, 2016.

[PAE08] Christopher Parisien, Charles H Anderson, and Chris Eliasmith. Solving the problem of negative synaptic weights in cortical models. *Neural computation*, 20(6):1473–1494, 2008.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[PKNY20] 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.

[PP18] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, page 774, 2018.

[PP21] Christian Pehle and Jens Egholm Pedersen. Norse - A deep learning library for spiking neural networks, January 2021. Documentation: https://norse.ai/docs/.

[PSRGSGLB20] Alberto Patiño-Saucedo, Horacio Rostro-Gonzalez, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. Event-driven implementation of deep spiking convolutional neural networks for supervised classification using the spinnaker neuromorphic platform. *Neural Networks*, 121:319–328, 2020.

[PVSDC20] Federico Paredes-Valles, Kirk Yannick Willehm Scheper, and Guido Cornelis Henricus Eugene De Croon. Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2051–2064, 2020.

[PWZ$^+$19] Zihan Pan, Jibin Wu, Malu Zhang, Haizhou Li, and Yansong Chua. Neural population coding for effective temporal classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[QGL$^+$20] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.

[RBB$^+$18] Oliver Rhodes, Petruţ A Bogdan, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R Lester, Mantas Mikaitis, Luis A Plana, Andrew GD Rowley, et al. spynnaker: a software package for running pynn simulations on spinnaker. *Frontiers in neuroscience*, 12:816, 2018.

[RDS$^+$15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RLH$^+$17] 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.

[RR20] Nitin Rathi and Kaushik Roy. Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658*, 2020.

[RT11] Edmund T Rolls and Alessandro Treves. The neuronal encoding of information in the brain. *Progress in neurobiology*, 95(3):448–490, 2011.

[SBB$^+$18] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. Hats: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1731–1740, 2018.

[SBG$^+$10] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950. IEEE, 2010.

[SBG19] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *Elife*, 8:e47314, 2019.

[SBMN17] Jeffrey M Shainline, Sonia M Buckley, Richard P Mirin, and Sae Woo Nam. Superconducting optoelectronic circuits for neuromorphic computing. *Physical Review Applied*, 7(3):034013, 2017.

[SGLB15] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Frontiers in neuroscience*, 9:481, 2015.

[SGN20] Marcel Stimberg, Dan FM Goodman, and Thomas Nowotny. Brian2genn: accelerating spiking neural network simulations with graphics hardware. *Scientific reports*, 10(1):1–12, 2020.

[SHY18] Changhyuck Sung, Hyunsang Hwang, and In Kyeong Yoo. Perspective: A review on memristive hardware for neuromorphic computation. *Journal of Applied Physics*, 124(15):151903, 2018.

[SJ20] Clemens JS Schaefer and Siddharth Joshi. Quantizing spiking neural networks with integers. In *International Conference on Neuromorphic Systems 2020*, pages 1–8, 2020.

[SJP$^+$20] Kyung Mee Song, Jae-Seung Jeong, Biao Pan, Xichao Zhang, Jing Xia, Sunkyung Cha, Tae-Eon Park, Kwangsu Kim, Simone Finizio, Jörg Raabe, et al. Skyrmion-based artificial synapses for neuromorphic computing. *Nature Electronics*, 3(3):148–155, 2020.

[SLBS20] Martino Sorbaro, Qian Liu, Massimo Bortone, and Sadique Sheik. Optimizing the energy consumption of spiking neural networks for neuromorphic applications. *Frontiers in neuroscience*, 14:662, 2020.

[SM19] Christoph Stöckl and Wolfgang Maass. Recognizing images with at most one spike per neuron. *arXiv preprint arXiv:2001.01682*, 2019.

[SM20] Christoph Stöckl and Wolfgang Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *arXiv preprint arXiv:2002.00860*, 2020.

[SM21] Christoph Stöckl and Wolfgang Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 3(3):230–238, 2021.

[SNG$^+$15] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[SNP$^+$15] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in neuroscience*, 9:222, 2015.

[SO18] Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, 31, 2018.

[SS14] Biswa Sengupta and Martin B Stemmler. Power consumption during neuronal computation. *Proceedings of the IEEE*, 102(5):738–750, 2014.

[SSSW08] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[SYW$^+$19] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TE16] Bryan Tripp and Chris Eliasmith. Function approximation in inhibitory networks. *Neural Networks*, 77:95–106, 2016.

[TGK+19] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019.

[TKPM21] Guangzhi Tang, Neelesh Kumar, Ioannis Polykretis, and Konstantinos P Michmizos. Biograd: Biologically plausible gradient-based learning for spiking neural networks. *arXiv preprint arXiv:2110.14092*, 2021.

[Tow91] Albert Towle. *Modern biology*. Holt McDougal, 1991.

[War18] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[WCZ+19] Jibin Wu, Yansong Chua, Malu Zhang, Qu Yang, Guoqi Li, and Haizhou Li. Deep spiking neural network with spike count based learning rule. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2019.

[WDL+18] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

[WS19] Pete Warden and Daniel Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[WXH+21] Jibin Wu, Chenglin Xu, Xiao Han, Daquan Zhou, Malu Zhang, Haizhou Li, and Kay Chen Tan. Progressive tandem learning for pattern recognition with deep spiking neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[ZM18] Nan Zheng and Pinaki Mazumder. Learning in memristor crossbar-based spiking neural networks through modulation of weight-dependent spike-timing-dependent plasticity. *IEEE Transactions on Nanotechnology*, 17(3):520–532, 2018.

[ZSG90] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at mit: Timit and beyond. *Speech communication*, 9(4):351–356, 1990.

[ZY18] Shanglin Zhou and Yuguo Yu. Synaptic ei balance underlies efficient neural coding. *Frontiers in Neuroscience*, 12:46, 2018.