Models and algorithms for implementing energy-efficient spiking neural networks on neuromorphic hardware at the edge

Manon Dampfhoffer

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : Nano électronique et Nano technologies
Unité de recherche : Spintronique et Technologie des Composants

Modèles et algorithmes pour l'implémentation de réseaux de neurones impulsionnels à faible consommation énergétique sur du matériel neuromorphique

Models and algorithms for implementing energy-efficient spiking neural networks on neuromorphic hardware at the edge

Présentée par :

Manon DAMPFHOFFER

Direction de thèse :

Lorena ANGHEL PROFESSEUR DES UNIVERSITES, Université Grenoble Alpes	Directrice de thèse
Alexandre VALENTIAN Ingénieur de recherche, Université Grenoble Alpes	Co-encadrant de thèse
Thomas MESQUIDA CEA Grenoble	Co-encadrant de thèse

Rapporteurs :

Timothée MASQUELIER
DIRECTEUR DE RECHERCHE, CNRS DELEGATION OCCITANIE OUEST
Benoît MIRAMOND
PROFESSEUR DES UNIVERSITES, UNIVERSITE COTE D'AZUR

Thèse soutenue publiquement le **4 septembre 2023**, devant le jury composé de :

Lorena ANGHEL PROFESSEURE DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Directrice de thèse
Timothée MASQUELIER DIRECTEUR DE RECHERCHE, CNRS DELEGATION OCCITANIE OUEST	Rapporteur
Benoît MIRAMOND PROFESSEUR DES UNIVERSITES, UNIVERSITE COTE D'AZUR	Rapporteur
Melika PAYVAND ASSISTANT PROFESSOR, Universität Zürich	Examinatrice
Pascal PERRIER PROFESSEUR DES UNIVERSITES, GRENOBLE INP	Président
Damien QUERLIOZ CHARGEE DE RECHERCHE HDR, CNRS DELEGATION ILE-DE-FRANCE SUD	Examinateur

Invités :

Thomas Mesquida
INGENIEUR DOCTEUR, Université Grenoble Alpes, CEA, List
Alexandre Valentian
INGENIEUR DOCTEUR, Université Grenoble Alpes, CEA, List

Abstract

Deep learning in Artificial Neural Networks (ANNs), a branch of Artificial Intelligence (AI), is considered a revolution in computing and is impacting every sectors of the economy. However, ANNs are very compute- and memory-intensive, which limits their integration into edge devices for embedded applications. Bio-inspired Spiking Neural Networks (SNNs) are promising energy-efficient alternatives to ANNs, and hence are good candidates for edge AI implementations using neuromorphic hardware. Indeed, SNNs encode the information using sparse temporal events (called spikes) instead of dense and high precision activations. However, the gap between the algorithmic development of SNNs on the one hand, and their hardware implementation on the other hand, makes it difficult to achieve truly efficient solutions. In this context, this thesis follows a hardware-aware approach to drive algorithmic developments of SNNs. In particular, models and algorithms are proposed for improving the accuracy and energy efficiency of SNNs, considering both digital and analog hardware implementations.

In the interests of comparing SNNs and ANNs implementations on dedicated neural network accelerators, a high-fidelity model of their energy efficiency is provided. In particular, it is found that spike sparsity plays a key role in the efficiency of SNNs. Consequently, a novel SNN model, SpikGRU, combining the accuracy of gated recurrent ANNs with a high spike sparsity, is proposed. In addition, the implementation of synaptic weights with analog non-volatile memories is considered to further increase the energy efficiency. With an adapted training methodology, SNNs are demonstrated to be very robust to these highly-quantized and noisy weights. A case study using resistive memories further validates the approach.

By promoting algorithm-hardware co-development, this work aims at paving the way for efficient neural network implementations at the edge.

Keywords: deep learning, artificial neural networks, spiking neural networks, neuromorphic hardware, neural network accelerators, non-volatile memories

Résumé

L'apprentissage profond dans les réseaux de neurones artificiels (ANNs), une branche de l'intelligence artificielle (IA), est considéré comme une révolution dans l'informatique et a un impact sur tous les secteurs de l'économie. Cependant, les ANNs sont très gourmands en ressources de calcul et en mémoire, ce qui limite leur intégration à la périphérie du réseau pour des applications embarquées. Les réseaux de neurones impulsionnels (SNNs) sont des alternatives prometteuses aux ANNs en termes d'efficacité énergétique et sont donc de bons candidats pour les implémentations IA embarquées utilisant du matériel neuromorphique. En effet, les SNNs encodent les informations en utilisant des événements temporels épars (appelés "spikes") au lieu d'activations denses et précises. Cependant, l'écart entre le développement algorithmique des SNNs d'une part, et leur implémentation matérielle d'autre part, rend difficile l'obtention de solutions réellement efficaces. Dans ce contexte, cette thèse suit une approche tenant compte du matériel pour conduire les développements algorithmiques des SNNs. En particulier, des modèles et des algorithmes sont proposés pour améliorer la précision et l'efficacité énergétique des SNNs, en considérant des implémentations matérielles numériques et analogiques.

Afin de comparer les implémentations des SNNs et des ANNs sur des accélérateurs de réseaux de neurones dédiés, un modèle de leur efficacité énergétique est fourni. En particulier, on constate que la parcimonie des activations joue un rôle clé dans l'efficacité des SNNs. Par conséquent, un nouveau modèle de SNN, SpikGRU, combinant la précision des ANNs récurrents à porte avec une parcimonie des activations, est proposé. En outre, l'implémentation des poids synaptiques utilisant des mémoires analogiques non volatiles est envisagée pour augmenter encore l'efficacité énergétique. Avec une méthodologie d'apprentissage adaptée, les SNNs se révèlent très robustes à ces poids hautement quantifiés et avec un haut niveau de bruit. Une étude de cas utilisant des mémoires résistives valide l'approche.

En encourageant le co-développement algorithme-matériel, ce travail vise à ouvrir la voie à des implémentations efficaces de réseaux de neurones embarqués.

Mots-clés: apprentissage profond, réseaux de neurones artificiels, réseaux de neurones impulsionnels, matériel neuromorphique, accélérateurs de réseaux de neurones, mémoires non-volatiles

List of publications

Journals:

- J. Minguet Lopez, T. Hirtzlin, M. Dampfhoffer, L. Grenouillet, L. Reganaz, G. Navarro, C. Carabasse, E. Vianello, T. Magis, D. Deleruyelle, M. Bocquet, J. M. Portal, F. Andrieu and G. Molas, OxRAM+OTS optimization for binarized neural network hardware implementation, *Semicond. Science Tech.* **2022**, 37, 014001, doi: 10.1088/1361-6641/ac31e2.

- J. Minguet Lopez, Q. Rafhay, M. Dampfhoffer, L. Reganaz, N. Castellani, V. Meli, S. Martin, L. Grenouillet, G. Navarro, T. Magis, C. Carabasse, T. Hirtzlin, E. Vianello, D. Deleruyelle, J. M. Portal, G. Molas and F. Andrieu, 1S1R optimization for high-frequency inference on Binary Spiking Neural Networks, *Advanced Electronic Materials* **2022**, 2200323, doi: 10.1002/aelm.202200323.

- M. Dampfhoffer, T. Mesquida, A. Valentian and L. Anghel, Backpropagation-Based Learning Techniques for Deep Spiking Neural Networks: A Survey, in *IEEE Transactions on Neural Networks and Learning Systems*, **2023**, doi: 10.1109/TNNLS.2023.3263008.

- M. Dampfhoffer, T. Mesquida, A. Valentian and L. Anghel, Are SNNs Really More Energy-Efficient Than ANNs? an In-Depth Hardware-Aware Study, in *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 3, pp. 731-741, June **2023**, doi: 10.1109/TETCI.2022.3214509.

Conferences:

- M. Dampfhoffer, T. Mesquida, A. Valentian and L. Anghel, Investigating Current-Based and Gating Approaches for Accurate and Energy-Efficient Spiking Recurrent Neural Networks. In: Pimenidis, E., Angelov, P., Jayne, C., Papaleonidas, A., Aydin, M. (eds) *Artificial Neural Networks and Machine Learning – ICANN 2022*. ICANN **2022**. Lecture Notes in Computer Science, vol 13531. Springer, Cham. doi: 10.1007/978-3-031-15934-3_30.

- M. Dampfhoffer, T. Mesquida, E. Hardy, A. Valentian and L. Anghel, Leveraging Sparsity with Spiking Recurrent Neural Networks for Energy-Efficient Keyword Spotting, *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Rhodes Island, Greece, **2023**, pp. 1-5, doi: 10.1109/ICASSP49357.2023.10097174.

- M. Dampfhoffer, J. Minguet Lopez, T. Mesquida, A. Valentian and L. Anghel, Improving the Robustness of Neural Networks to Noisy Multi-Level Non-Volatile Memory-based Synapses, *International Joint Conference on Neural Networks (IJCNN)*, Gold Coast, Australia, **2023**, pp. 1-8, doi: 10.1109/IJCNN54540.2023.10191804.

- J. Minguet Lopez, M. Dampfhoffer, T. Hirtzlin, L. Reganaz, L. Grenouillet, G. Navarro, M. Bernard, T. Magis, C. Carabasse, N. Castellani, V.Meli, E. Vianello, D. Deleruyelle, M. Bocquet, J. M. Portal, G. Molas and F. Andrieu, "1S1R Sub-Threshold Operation in Crossbar Arrays for Neural Networks Hardware Implementation," *2023 30th International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*, Kraków, Poland, **2023**, pp. 1-6, doi: 10.23919/MIXDES58562.2023.10203226.

- T. Mesquida, M. Dampfhoffer, T. Dalgaty, P. Vivet, A. Sironi and C. Posch, "G2N2: Lightweight Event Stream Classification with GRU Graph Neural Networks", *34th British Machine Vision Conference 2023, BMVC 2023*, Aberdeen, UK, November 20-24, **2023**.

Acknowledgments

This thesis would not have been as it is without the support of the people around me. In particular, I want to express my gratitude to my thesis directors, Lorena Anghel and Alexandre Valentian for giving me the opportunity to do this thesis and for their encouragement. I want to specially thank my co-supervisor Thomas Mesquida for the endless discussions on digital design and algorithms and for having shared his knowledge and ideas with me.

I want to thank the University Grenoble Alpes and MIAI institute, who funded this thesis. I also want to thank the people at DSCIN for their support, and in particular people in the LSTA laboratory for accompanying me during this thesis. I want to express my gratitude to people from the neuro team, in particular Ivan Miro Panades, François Rummens, for their helpful advice. I would also like to thank Antoine Héraud, Stéphane Burel, Michele Martemucci, Yannick Malot for having shared a part of the PhD journey. I would like to specially thank Adrian Evans for his support and advice, for helping the students, and whose scientific rigor inspired me. I want to thank Pascal Vivet for following my thesis work and his advice.

I want to express my gratitude to people from the LIIM laboratory who followed my thesis project, such as Marielle Malfante, Marina Reyboz and Thomas Dalgaty. I also want to thank Diego Puschini, Olivier Antoni, Anca Molnos and Romain Lemaire, for having shared with me their projects. I also want to thank Johannes Thiele from LIAE laboratory for passing on his vision on spiking neural networks.

I want to thank all the people in Spintec laboratory for sharing their interest in physics and AI, in particular Lucian Prejbeanu, Philippe Talatchian, Ursula Ebels, Kamal Danouchi, Guillaume Prenat and Gregory Di Pendina.

I want to express my gratitude to people from CEA-Leti for sharing their passion for analog devices and design: Gabriel Molas, Elisa Vianello, Tifenn Hirtzlin and, in particular, Joel Minguet Lopez for introducing me to non-volatile memories, and Emmanuel Hardy for sharing his experience on low-power design for keyword spotting.

I also want to thank the members of the jury, Timothée Masquelier, Benoît Miramond, Damien Querlioz, Melika Payvand and Pascal Perrier, for having accepted to examine my work.

Finally, I want to express my gratitude to my loved ones, for their support throughout this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

1S1R	1 Selector 1 Resistor
1T1R	1 Transistor 1 Resistor
AC	ACcumulate
Adapt-LIF	Adaptive Leaky Integrate-and-Fire
AER	Address Event Representation
ANN	Artificial Neural Network
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
BN	Batch Normalization
BPTT	BackPropagation Through Time
BSNN	Binary Spiking Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
Cuba-LIF	Current-based Leaky Integrate-and-Fire
DAS	Dynamic Audio Sensor
DRAM	Dynamic Random Access Memory
DVS	Dynamic Vision Sensor
FC	Fully Connected
FPGA	Field Programmable Gate Array
GLB	GLobal Buffer
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
iact	Input activation
IF	Integrate-and-Fire
IMC	In-Memory Computing
KWS	KeyWord Spotting
LIF	Leaky Integrate-and-Fire
LSTM	Long Short-Term Memory
MAC	Multiply-and-ACcumulate
NMC	Near-Memory Computing
NVM	Non-Volatile Memory
oact	Output activation
PE	Processing Element
psum	Partial sum
ReLU	Rectified Linear Unit
RF	Reuse Factor
RNN	Recurrent Neural Network
RRAM	Resistive Random Access Memory
SCNN	Spiking Convolutional Neural Network

SNN	Spiking Neural Network
SpikGRU	Spiking Gated Recurrent Unit
SRAM	Static Random Access Memory
SRNN	Spiking Recurrent Neural Network
STDP	Spike-Timing-Dependent Plasticity
tanh	Hyperbolic tangent

Chapter 1

Introduction

Deep learning in Artificial Neural Networks (ANNs), a branch of artificial intelligence (AI), is considered a revolution in computing and is impacting every sectors of the economy. Indeed, ANNs can now solve difficult tasks at the human level and beyond, and are a game changer in many fields, such as transportation, health, or industry. For instance, ANNs are used in many applications such as image recognition (Krizhevsky et al., 2009), object detection (Girshick et al., 2014), speech recognition (Hinton et al., 2012), medical diagnosis (Esteva et al., 2017), game playing (Silver et al., 2016), etc. Nevertheless, ANNs are very compute- and memory-intensive and are responsible, in part, for the growth of the CO2 emissions of the Cloud (Li et al., 2016). Therefore, directly integrating AI algorithms into edge devices can allow to decrease data transfer between the devices and the Cloud, hence reducing energy consumption, but also latency, dependency on connectivity, as well as improving security and privacy. To this end, research is increasingly moving towards efficient hardware implementations of ANNs on dedicated accelerators, which could be deployed at the edge. While an increased speed and energy efficiency have been achieved, further gains could result from the combination of specialized hardware and more efficient algorithms.

By more closely mimicking the brain, Spiking Neural Networks (SNNs) appear to be energy-efficient alternatives to ANNs. In the brain, neurons use electrical pulses to transmit information through the synapses in a sparse and asynchronous manner. Similarly, SNNs encode the information using sparse temporal events (called spikes) instead of dense and high precision activations. These input spikes are integrated through time in the membrane potential of neurons, the latter firing when reaching its threshold, following the Integrate-and-Fire (IF) dynamics (Lapicque, 1907). In addition, their ability to exploit spatio-temporal information makes them attractive for various applications, and in particular for processing event data produced by low-power dynamic sensors (Lichtsteiner et al., 2008).

SNNs present many advantages for efficient implementation on so-called neuromorphic hardware (Mead, 1990). Indeed, while ANNs process the high precision information in a one-shot fashion using matrix multiplications, information in SNN is coded in a binary signal distributed over time. The use of spikes allows replacing costly multiply-accumulate (MAC) operations in ANNs by simpler accumulate (AC) operations, which consume less energy and occupy less area (Horowitz, 2014). Moreover, the high spike sparsity can be leveraged efficiently in event-based implementations (Merolla et al., 2014; Davies et al., 2018; Moradi et al., 2018). Therefore, SNNs are considered good candidates for edge AI implementations.

However, the gap between the algorithmic development of SNNs on the one hand, and their hardware implementation on the other hand, makes it difficult to achieve

truly efficient solutions. Indeed, there is currently no general model allowing to estimate the energy consumption of SNNs on neuromorphic hardware, making it difficult to ensure that they are actually more efficient than ANNs. Moreover, analog hardware implementations of SNNs, in particular using emerging non-volatile memories (NVMs) to encode synaptic weights, can achieve significant gains compared to fully-digital implementations (Hung et al., 2022). However, analog systems are prone to variability, inducing the occurrence of errors, which can significantly degrade the accuracy of the system (Higuchi et al., 2022; Yan et al., 2023). Therefore, a hardware-algorithm co-development strategy is needed in order to obtain accurate and energy-efficient solutions for edge AI applications.

In this thesis, we first propose a high-fidelity model of the dynamic energy consumption of SNNs and ANNs, in the interests of comparing their implementation on dedicated neural network accelerators. We provide lower and upper bounds on the relative efficiency of ANNs and SNNs, as well as a case study using state-of-the-art neural network accelerators. In particular, we find that spike sparsity plays a key role in the efficiency of SNNs. Unfortunately, we show that SNN algorithms based on convolutional topologies for processing static data do not reach a sufficient spike sparsity to compete with efficient ANN implementations.

Consequently, we propose a novel SNN model, SpikGRU, combining the accuracy of gated recurrent ANNs with a high spike sparsity, for processing spatio-temporal data. SpikGRU is compared with various recurrent SNN and ANN models on several spiking and non-spiking data, using speech recognition tasks. Furthermore, with the example of SpikGRU, we show that sparsity in SNNs can be further leveraged by optimizing the activity of neurons through gradient descent, or by using sparse spiking input data. In addition, we demonstrate that SpikGRU can allow higher energy efficiency than ANN equivalents on a dedicated neuromorphic hardware implementation, while being as accurate.

Furthermore, we consider the implementation of synaptic weights with analog NVMs as a solution to improve the efficiency of both ANN and SNN implementations. We present a fault model applicable to all kind of single- and multi-level NVMs and an adapted training methodology. ANNs and SNNs with convolutional and recurrent topologies are demonstrated to be robust to errors in these highly-quantized and noisy weights. A case study using resistive memories further validates the approach.

This thesis is organized as follows:

- In Chapter 2, the background on SNN algorithms and hardware implementations is presented, focusing on strategies to improve their accuracy and energy efficiency.
- In Chapter 3, a model of the dynamic energy consumption of SNNs and ANNs on dedicated neural network accelerators is proposed, providing guidelines for improving SNN algorithms and hardware.
- In Chapter 4, a novel gated recurrent SNN model (SpikGRU) is introduced and its high accuracy and energy efficiency are demonstrated on several spoken word recognition tasks.

- In Chapter 5, SNNs and ANNs robustness to errors in highly-quantized and noisy weights is studied, in the objective of implementing the synaptic weights with NVMs. A case study with resistive memories is considered.
- In Chapter 6, a summary of the contributions of the thesis and the perspectives are presented.

Chapter 2

Background and State of the Art

Spiking Neural Networks (SNNs) are studied from the perspective of neurosciences, machine learning and neuromorphic hardware, with different objectives, and hence are characterized by an important heterogeneity. Various approaches, with varying degrees of biological plausibility, have been proposed to model and train SNNs. Moreover, different types of neuromorphic hardware, from digital to analog, have been considered to efficiently implement SNNs. This chapter provides a general overview of the state-of-the-art on SNNs, with a focus on the background that was used for this work.

This chapter is organized as follows:

- In Section 2.1, models and implementations of SNNs are presented, including models of neurons and synapses, information encoding with spikes, software and hardware implementations of SNNs, and applications for SNNs.
- In Section 2.2, training strategies for SNNs are reviewed, from biologically-inspired learning rules to high performance backpropagation-based training.
- In Section 2.3, strategies to enhance accuracy and efficiency (such as latency and spike sparsity) of SNNs are presented. Special attention is given to the trade-off between accuracy and latency, as well as the estimation of the energy efficiency.

Part of the state of the art presented in this chapter has been published in Dampfhoffer et al., 2023c.

2.1 Models and Implementations of Spiking Neural Networks

2.1.1 Models and Coding Strategies

SNN Models

The basic ANNs and SNNs units in a neural networks are shown in Fig.2.1. In ANNs, the output of a neuron is a function defined as:

$$y_i = \varphi(\sum_j x_j w_{ij} + b_i) \tag{2.1}$$

where y_i is the output activation of neuron i, b_i is the bias of neuron i, x_j is the input activation from presynaptic neuron j, and w_{ij} is the synaptic weight between neurons

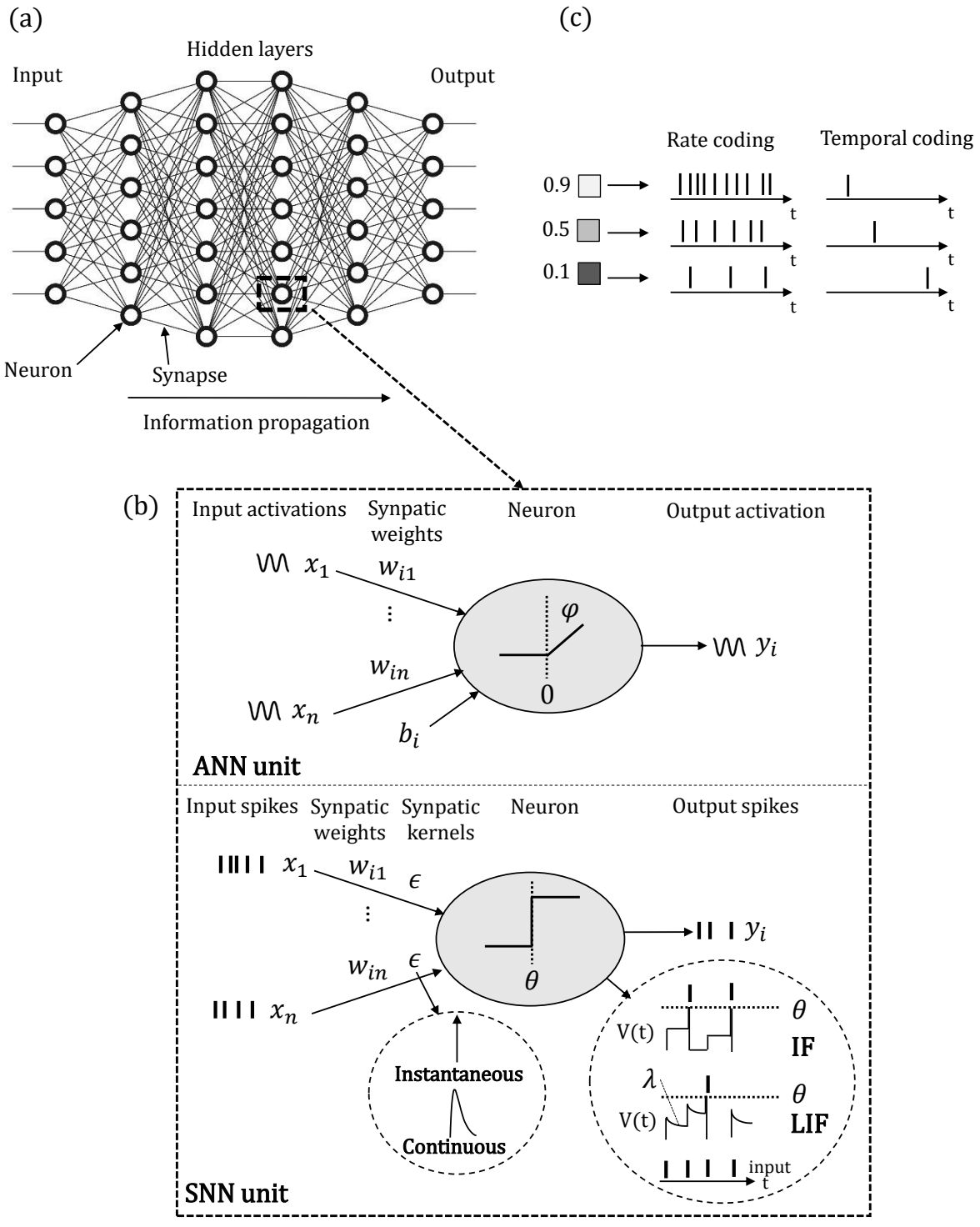


FIGURE 2.1: **Models and coding strategies for Spiking Neural Networks.** (a) Feedforward fully-connected neural network. (b) ANN and SNN neuron and synapse models. (c) Input encoding: example of pixel-to-spike conversion with a rate coding or temporal (latency) coding.

i and j. φ is an activation function, such as the Rectified linear unit (ReLU). While in ANNs the information propagates synchronously on a layer-by-layer basis, the information processing in SNNs is asynchronous and in a depth-first manner. Indeed, neurons in a layer fire spikes without waiting for other neurons in the same layer to fire. Moreover, due to the temporal dynamics of the neurons, SNNs necessarily operate in the spatio-temporal domain, while standard ANNs operate only in the spatial domain.

The most popular neuron model for SNNs is the Leaky Integrate-and-Fire model (LIF) Lapicque, 1907; Gerstner et al., 2014. More biogically-plausible neuron models exist, such as Hodgkin et al., 1952; Izhikevich, 2003, but have not yet demonstrated superior performance than the simple LIF model for deep learning applications and are computationally much more expensive.

In the LIF model, similar to biological neurons, the neuron integrates the weighted input spikes into its membrane potential. When the latter reaches its threshold, the neuron fires an output spike and the membrane potential is reset. The membrane potential $V_i(t)$ of the neuron i in the LIF model is described as :

$$\lambda \frac{\mathrm{d}V_i}{\mathrm{d}t} = -V_i + \sum_j w_{ij} \sum_k \epsilon(t - t_{jk}) \tag{2.2}$$

$$V_i(t) = V_{reset}, \text{ if } V_i(t) \geq v_{th} \tag{2.3}$$

where λ is the membrane time constant, w_{ij} is the synaptic weight from neuron j to i, $\epsilon(.)$ is the synaptic kernel, t_{jk} is the k^{th} spike of the input neuron j, V_{reset} is the reset membrane potential and v_{th} is the membrane potential threshold. This model describes many types of LIF variants. For instance, in the non-leaky version of the LIF neuron (IF), the membrane potential does not decay over time, and hence remains constant in between spikes. This is obtained by removing the $-V_i$ and setting λ to 1 in equation 2.2. Moreover, the synapse model is defined by the kernel function $\epsilon(.)$, corresponding to the response of the membrane potential to the presynaptic spike. The synapse can be instantaneous (Dirac kernel function) or continuous (e.g. linear, exponential, or alpha kernel functions), allowing to model various synaptic behaviors. In particular, the combination of LIF neurons with exponential continuous synapses is also called Current-based LIF (Cuba-LIF). Note that other variants exist that are not described by this model, such as the Adaptive LIF (Adapt-LIF) (e.g. in Bellec et al., 2018b; Yin et al., 2021), that uses an adaptive threshold described with temporal dynamics (the threshold is increased after each spike fired and decays exponentially with time).

The inference phase of SNNs is usually discretized in timesteps in order to simulate their spatio-temporal dynamics. Each timestep corresponding to a forward pass in the network, the number of timestep can allow to estimate the future latency of the SNN inference in hardware. In this context, an iterative version of the LIF model (as in Wu et al., 2018) is used, similar to the description of Recurrent Neural Networks (RNNs). A common iterative description of the LIF with an instantaneous synapse in a deep SNN is:

$$V_i^l(t) = \beta V_i^l(t-1) + \sum_j w_{ij} s_j^{l-1}(t) + b_i - v_{th} s_i^l(t-1) \tag{2.4}$$

$$s_i^l(t) = H[V_i^l(t) - v_{th}] \tag{2.5}$$

$s_i^l(t)$ denotes the output spikes of neuron i from layer l at time t. Spike firing happens when the membrane potential is superior to the threshold v_{th}, which corresponds to the Heaviside step function H. In this description, the threshold v_{th} is substracted from the membrane potential after each spike. This corresponds to a "soft" version of the reset (Han et al., 2020b) instead the "hard" version of the reset described in equation 2.2. The parameters of the models are W and b, respectively weights and biases, and the time constant β.

Information Encoding With Spikes

While ANNs use static high precision activations per neuron per inference, SNNs use one or several binary spikes to code the activation. There are several coding strategies based on the spike rate, timing, rank, phase, etc. (for a review see Auge et al., 2021), but the majority of works in deep SNNs use either the spike rate or the spike timing. The rate coding strategy uses several spikes to represent one unit of information while in temporal coding, the information is carried by individual spike times. Note that, in order to be efficient in neuromorphic hardware, the coding strategy should use a minimum number of spikes, as the energy consumption is strongly correlated to the spiking activity. Moreover, the choice of the coding strategy is associated with the learning strategy (see Section 2.2).

The coding strategy must consider both the encoding of the input to the network and the decoding of the output. Indeed, to process real-valued data, such as pixels for images, these values can be converted into spikes in order to be processed by the SNN. Data can also be already in the form of spikes that can be fed directly to the SNN without pre-processing. This is the case for neuromorphic sensors, such as event cameras (Lichtsteiner et al., 2008) or artificial cochleas (Chan et al., 2007). Fig.2.1 represents a typical pixel-to-spike conversion in rate and time. The rate-based strategy matches each pixel intensity with a firing rate, using a probabilistic sampling (generally Poisson) to generate the spike trains: the higher the pixel value, the higher the firing rate of the corresponding input. A simple time-based strategy, also called latency coding, consists in associating the pixel intensity with the latency of a single spike. In that case, the latency is inversely proportional to the pixel intensity: earlier spikes encode higher values and later spikes encode lower values. More recently, Stanojevic et al., 2022 have used a linear latency coding (where the spike time is defined as the difference between a maximum predefined time and the ANN equivalent activation), also preserving this relationship (earlier spikes correspond to higher activations).

Decoding the output in a classification task consists in determining the most activated neuron in the output layer, each neuron being associated with a class. With rate coding, this can be done by using the highest spike rate, or the highest membrane potential value in non-spiking output neurons. With temporal coding, a solution called Time-To-First-Spike (TTFS) consist in using the first spike fired by one of the output neurons.

Temporal codes are supposed to be sparse and can have a lower latency (e.g. when the TTFS decoding is used). However, temporal coding may require high temporal resolution because each spike carries important information, which may be difficult to implement efficiently in neuromorphic hardware.

2.1.2 Software and Hardware Implementations

Off-Chip vs. On-Chip Training

Training of ANNs and SNNs is said "on-chip" or "off-chip", whether the training is performed on the chip that will also be used for the inference, or if the training is performed on a different computer architecture. The advantage of the latter is the possibility to train the networks with high-performance algorithms and computing architecture. However, in this case, the training cannot be performed once the network has been deployed on the chip. On the opposite, on-chip training could not only allow more efficient training of neural networks, but also could allow the system to continue to learn through its life-time using incremental learning techniques (such as Solinas. et al., 2021). Nevertheless, current high performance training algorithms, such as backpropagation, are very costly to implement on chip (Bengio et al., 2016). Therefore, if the system is not meant to be trained on-chip, using off-chip training techniques has allowed to reach the best performance so far. Hence, in the interest of maximizing accuracy and efficiency of SNNs only during the inference phase, off-chip training techniques have been used in this work.

Software Simulations of SNNs

Following the off-chip training strategy, SNNs can be simulated and trained on a general computer. On the one hand, software frameworks (such as Brian2, from Stimberg et al., 2019) can allow to simulate small-scale SNNs with high biological fidelity, described with differential equations. However, when training larger architectures for deep learning applications, these simulations are too costly. On the other hand, large-scale SNNs are usually trained using similar frameworks as ANNs, such as Pytorch (Paszke et al., 2019). These frameworks allow to benefit from automatic differentiation as well as a high parallelization of the computations accelerated with GPUs. In addition, specific frameworks based on Pytorch (such as SpykeTorch from Mozafari et al., 2019a or SpikingJelly from Fang et al., 2020b) have been developed to ease the training of SNNs, implementing various neuron models and learning rules.

Large-scale SNNs are often simulated with timesteps in a similar way as RNNs (see Section 2.2), and hence the simulation time scales with the number of timesteps. This incurs higher training time (and resources utilization) compared to the training of an ANN with the same topology. To mitigate this problem, some frameworks (such as SpikingJelly from Fang et al., 2020b) have targeted the acceleration of SNN training.

This thesis focuses on large-scale SNNs for deep learning applications. The Pytorch framework (Paszke et al., 2019) was used for all the simulations, rather than frameworks dedicated to SNNs, to facilitate the development of customed algorithms.

Implementations of SNNs on Neuromorphic Hardware

Neural network inference on general-purpose processors (CPUs or GPUs) is inefficient (see Fig. 2.2). Indeed, the separation of the processor and the external memory causes high data transfer between both units, leading to high energy consumption and latency, also called the von Neumann bottleneck or "memory wall" (Horowitz, 2014). This trend is exacerbated in neural networks as they rely on many memory accesses

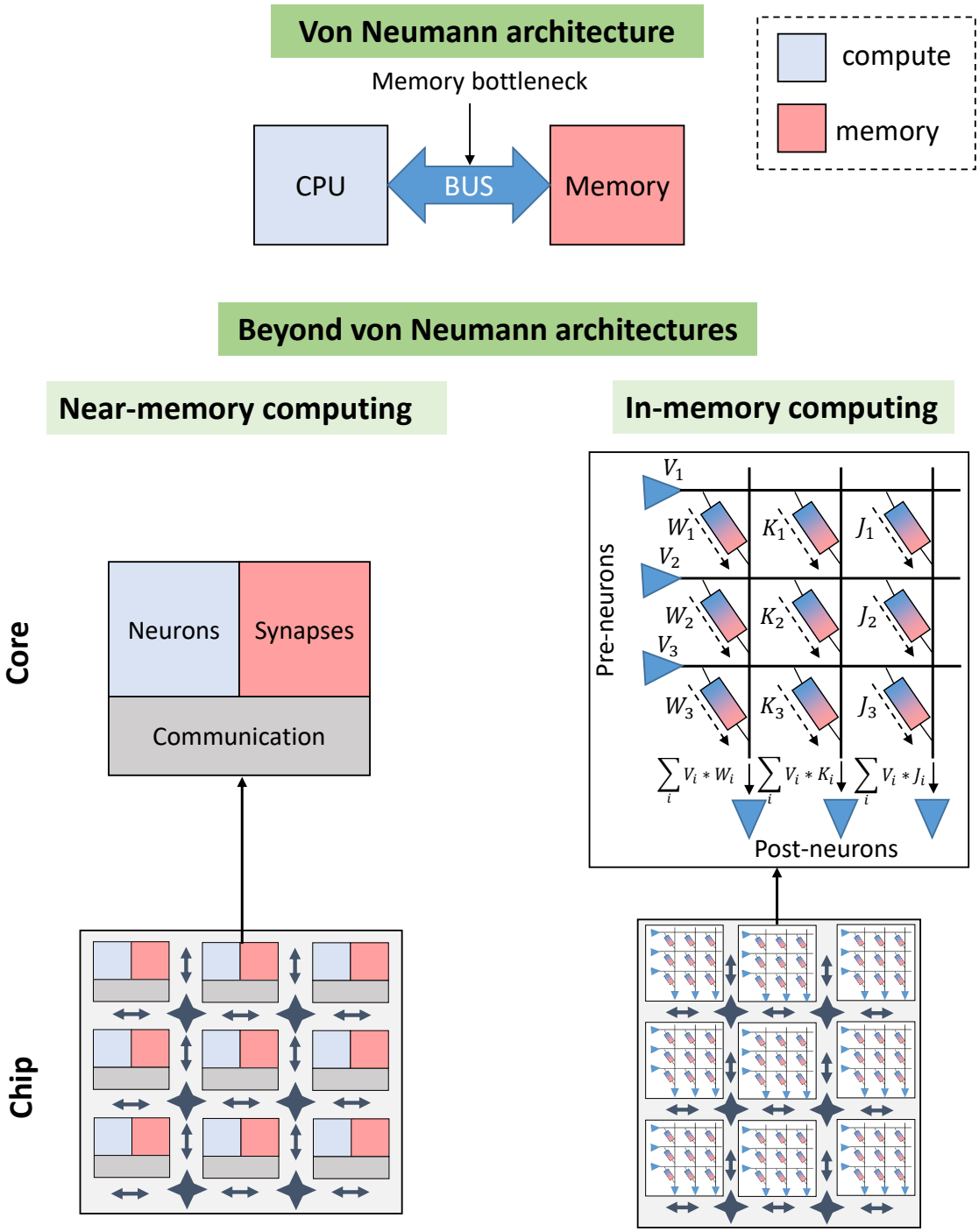


FIGURE 2.2: **Hardware implementations of Spiking Neural Networks.** Traditional computer architectures, such as Central or Graphics Processing Units (CPUs, GPUs), are based on the von Neumann paradigm, where compute and memory are physically separated, causing a memory bottleneck. In beyond von Neumann architectures, compute and memory are co-localized inside a core. A chip is composed of several cores communicating using a network-on-chip (NOC). Among the architectures that are used in efficient neural networks accelerators, near- and in-memory computing (NMC, IMC) can be distinguished. In NMC, although compute and memory units are close, they are still separated. In IMC, part of the computation (multiply and accumulate operations) is performed inside the memory, using the physical properties of the devices.

associated with computations, in particular due to matrix multiplications of neuronal activations and weights. Therefore, researchers are focusing on designing energy-efficient "beyond von Neumann" hardware for accelerating the inference (or training) of neural networks. Unlike traditional von Neumann processor architectures, these spatial architectures (as opposed to temporal architectures such as CPUs of GPUs), also called "near-memory computing", bring memory and compute spatially close together (see Fig. 2.2). In addition, neuromorphic computing was introduced very early by Mead, 1990 with the objective of mimicking some aspects of biological neural systems to obtain more efficient computer architectures, in particular using analog hardware. Indeed, in biological neural systems, neurons and synapses (thus compute and memory) are not physically separated. Moreover, biological systems are naturally analog rather than digital.

The combination of dedicated hardware and algorithm could be the key to efficient neural network implementations. In particular, SNN implementations on neuromorphic hardware promise interesting gains. Indeed, due to the use of spikes, SNNs require only accumulate (AC) operations instead of multiply-and-accumulate (MAC) operations between neuronal activations and weights. Moreover, the spike sparsity can be inherently exploited in event-based neuromorphic hardware.

In this context, Application Specific Integrated Circuits (ASICs), such as Truenorth from IBM (Merolla et al., 2014), Loihi from Intel (Davies et al., 2018), DYNAPs from ETH Zurich (Moradi et al., 2018), and Field Programmable Gate Arrays (FPGAs) implementations, such as Mostafa et al., 2017; Corradi et al., 2021, have been proposed. Among large scale digital architectures, TrueNorth (Merolla et al., 2014) is composed of 1 million neurons and 256 million synapses on 4096 cores, while the Loihi chip (Davies et al., 2018) has reached 8 million neurons and 8 billion synapses. Most of SNN implementations, such as TrueNorth, Loihi and DYNAPs, communicate spikes using the Address Event Representation (AER) protocol (Boahen, 2000). In AER, a spike (i.e. an event) is encoded in a packet, containing the address of the source neuron, that is sent to the destination neuron in real time. The AER encoding allow to easily leverage the spike sparsity in SNNs, as null activations are not stored, and operations are triggered by the arrival of an event in an asynchronous manner. In addition, as connectivity is limited (in particular in two-dimensional design), large-scale chips require network-on-chip (NOCs) to manage spike communication between the different cores of a chip. Note that, although most accelerators focus on inference rather than learning, some chips, such as Loihi (Davies et al., 2018), include on-chip training capabilities. In particular, the version 2 of Loihi (Orchard et al., 2021) offers increased learning capabilities and programmability of the neuron model. Moreover, Loihi 2 is proposed with an open-source neuromorphic computing framework (Lava) that allows users to map SNN algorithms to neuromophic platforms.

Although traditional neural networks accelerators are based on fully-digital architectures (such as TrueNorth or Loihi), analog implementations have gained interest. Indeed, some of the essential operations of neural networks, such as the leak of neurons in SNNs or the multiplications and accumulations, can be realized very efficiently using the natural dynamics of physical systems, as shown by Mead, 1990. For instance, a capacitance can be used for implementing the leaky integration behavior of the membrane potential of analog neurons (Joubert et al., 2012). Moreover, synaptic weights can be implemented using analog emerging Non-Volatile Memory (NVMs)

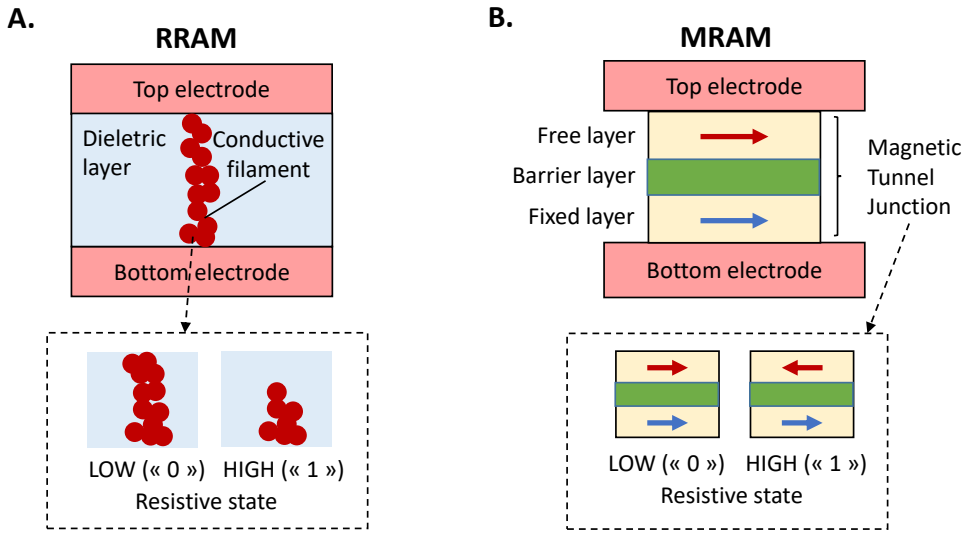


FIGURE 2.3: **Non-Volatile Memories** can be used to encode synaptic weights. **A.** Resistive Random Access Memory (RRAM) device. The resistance of the dielectric layer depends on the formation of the conductive filament. Low (resp. high) resistive state encodes the "0" (resp. "1"). **B.** Magnetic Random Access Memory (MRAM) device. The resistance of the magnetic tunnel junction depends on the orientation of the magnetization of the free layer: parallel (resp. anti-parallel) corresponds to a low (resp. high) resistive state and encodes the "0" (resp. "1").

devices, instead of static and dynamic random-access memories (SRAMs and DRAMs) which are typically used in digital neural network accelerators. NVMs retain the information even if the power supply is turned off, allowing to remove the dependency to an external memory and thus reducing the energy consumption of the system. These devices allow to encode one bit of information depending on the programmed state. For instance, in a resistive NVM, the resistance of the device represents the value to be stored, and can be at state "high" (representing 1) or "low" (representing 0). Moreover, multi-level programming strategies in NVMs allow more than one bit of information to be stored in a single NVM device, by encoding multiple non-volatile states in the memory. Multi-level implementations allow to increase the memory density, which is important for implementing large neural networks.

Various technologies of emerging NVMs are good candidates for such implementations (Ielmini et al., 2019), such as resistive RAMs (RRAMs), magnetic RAMs (MRAMs), phase-change RAMs (PCRAMs) or ferroelectric RAMs (FeRAMs) (see Fig. 2.3). For instance, Valentian et al., 2019 have demonstrated high energy efficiency with a SNN implementation combining analog neurons and RRAM-based synapses. Besides, Moradi et al., 2018 have realized an efficient combination of fully asynchronous digital communication and hybrid analog/digital circuits for synapses and neurons. Besides, NVMs can enable efficient in-memory computing (IMC), also called processing-in-memory (PIM), which is a promising alternative to reduce data movement according to Yang et al., 2019a (see Fig. 2.2). In such systems, the matrix-vector multiplication is directly performed using the physical properties of the analog devices in a highly-parallel fashion, as demonstrated in Joshi et al., 2020; Amrouch et al., 2021; Jung et al., 2022; Wan et al., 2022. However, device variability and non-ideality of analog circuits impose

constraints on the network architecture and can reduce the accuracy. Therefore, addressing these non-idealities is one of the main challenges for analog neural network implementations.

Note that, in this thesis, the term "real-valued" or "high precision" activations is used for describing ANN activations, as opposed to spiking activations. However, the precision of ANN activations can vary depending on the choice of hardware implementation.

2.1.3 Applications and Datasets

In principle, SNNs can be used with any topologies (such as convolutional, recurrent, etc.) for any deep learning applications, as ANNs. Many works (for instance Srinivasan et al., 2019; Han et al., 2020b; Zheng et al., 2021; Zhou et al., 2021; Fang et al., 2021a; Deng et al., 2022) have applied SNNs to image classification tasks, due to the popularity of datasets such as MNIST (Lecun et al., 1998), CIFAR (Krizhevsky et al., 2009) or ImageNet (Deng et al., 2009) for benchmarking purposes. However, SNNs have also been applied to other tasks, such as speech recognition (Bellec et al., 2018a; Wu et al., 2020), autonomous driving (Zhou et al., 2020; Viale et al., 2021), brain computer interfaces (Kasabov, 2014; Kumarasinghe et al., 2021), etc.

An interesting field of applications for SNNs is neuromorphic sensors, such as Dynamic Vision Sensors (DVS), also called event cameras (Lichtsteiner et al., 2008), and Dynamic Audio Sensors (DAS), also called silicon cochlea (Chan et al., 2007). These bio-inspired sensors output data directly in the form of spikes (sometimes with a polarity). For instance, in event cameras, pixels are sensitive to local changes in intensity, and asynchronously fire a spike, if a change in brightness occurs, or remain silent otherwise. This is completely different from traditional cameras producing frames at a given frame rate. Therefore, dynamic sensors can have a higher dynamic range and temporal resolution than conventional frame-based sensors (Lichtsteiner et al., 2008). In addition, these event-based sensors are particularly adapted to be processed by event-based SNNs, thus benefiting from their sparsity and asynchronous behavior. Different datasets produced by dynamic sensors have been used with SNNs. On the one hand, some are directly captured with the dynamic sensor, such as DVSGesture from Amir et al., 2017 for visual gesture recognition. On the other hand, other datasets were captured with a standard sensor and then converted to the neuromorphic domain. For instance, N-MNIST from Orchard et al., 2015 and CIFAR-10-DVS from Li et al., 2017 were created by showing the images of the respective MNIST (Lecun et al., 1998) and CIFAR-10 (Krizhevsky et al., 2009) datasets to the DVS. DASDIGITS from Anumula et al., 2018a was created by recording with the DAS the spoken digits from the TIDIGIT dataset (Leonard et al., 1993). Besides, other neuromorphic datasets have been generated using algorithms simulating the characteristics of dynamic sensors, such as the Spiking Heidelberg Dataset (SHD) from Cramer et al., 2020 for spoken digit or word recognition. Many works, such as Wu et al., 2019b; Kim et al., 2020; Zheng et al., 2021; Fang et al., 2021b; Deng et al., 2022, have benchmarked their SNN training strategy using DVSGesture or CIFAR-10-DVS, while Yin et al., 2020; Cramer et al., 2020; Perez-Nieves et al., 2021; Yin et al., 2021 have used the SHD dataset. In addition, Amir et al., 2017 shows an efficient hardware implementation of a spiking convolutional network

on TrueNorth yielding 96.5% accuracy on DVSGesture with a latency of 105ms and consuming less than 200mW.

2.2 Training Spiking Neural Networks

Learning is the process by which the parameters of the neural networks (e.g. weights, biases, time constants of neurons, etc.) are determined such that the network performs a specific task. The learning can be performed in a supervised way, if the data are labelled (meaning that the expected outputs are known), or in an unsupervised way. In the case of the supervised learning with gradient descent, a cost metric is defined as a function of the error between the desired and actual outputs. The learning process consists of tuning the parameters such that the cost function is minimized. In ANNs, the backpropagation algorithm is used to calculate the gradients of the cost function with respect to each synaptic weight in order to perform synaptic weights updates. The calculation starts from the last layer of the network and proceeds backwards layer by layer.

Various methods have been proposed to train SNNs in supervised or unsupervised manners. As proposed by Zhang et al., 2022, the techniques to train SNNs can be broadly classified into two main categories (see Fig. 2.4): (1) biologically-inspired learning rules and backpropagation-free techniques, and (2) backpropagation-based training strategies. While the first category tends to higher biological plausibility, the second category targets higher performance for deep learning applications.

2.2.1 Biologically-Inspired and Backpropagation-Free Learning Rules

Spike-Timing-Dependent Plasticity

The Spike-Timing-Dependent Plasticity (STDP) rule is an unsupervised local learning rule demonstrated by neurobiologists and is inspired by chemical mechanisms with computation capability existing in nervous systems Dan et al., 2006. The synapse's weight is strengthened (Long Term Potentiation) or weakened (Long Term Depression) depending on the relative timing of the spikes of the pre and postsynaptic neurons. The STDP is local in time and in space, and therefore is attractive for on-chip learning in neuromorphic hardware. However, due to its locality, STDP does not provide a global optimization scheme and the neural networks must be trained layer by layer. Therefore, the accuracy attainable with this rule is limited.

Kheradpisheh et al., 2018; Lee et al., 2018; Mozafari et al., 2018; Mozafari et al., 2019b; Srinivasan et al., 2019 have built multi-layer SNNs trained with STDP in a layer-wise manner and added some form of supervision to the training to improve its accuracy. Kheradpisheh et al., 2018 pre-process the input data by applying Difference-of-Gaussian filters to facilitate the feature extraction, followed by a spiking Convolutional Neural Network (CNN) trained with STDP. Next, the output was processed with Support Vector Machine (SVM) classifier to further perform the MNIST classification task

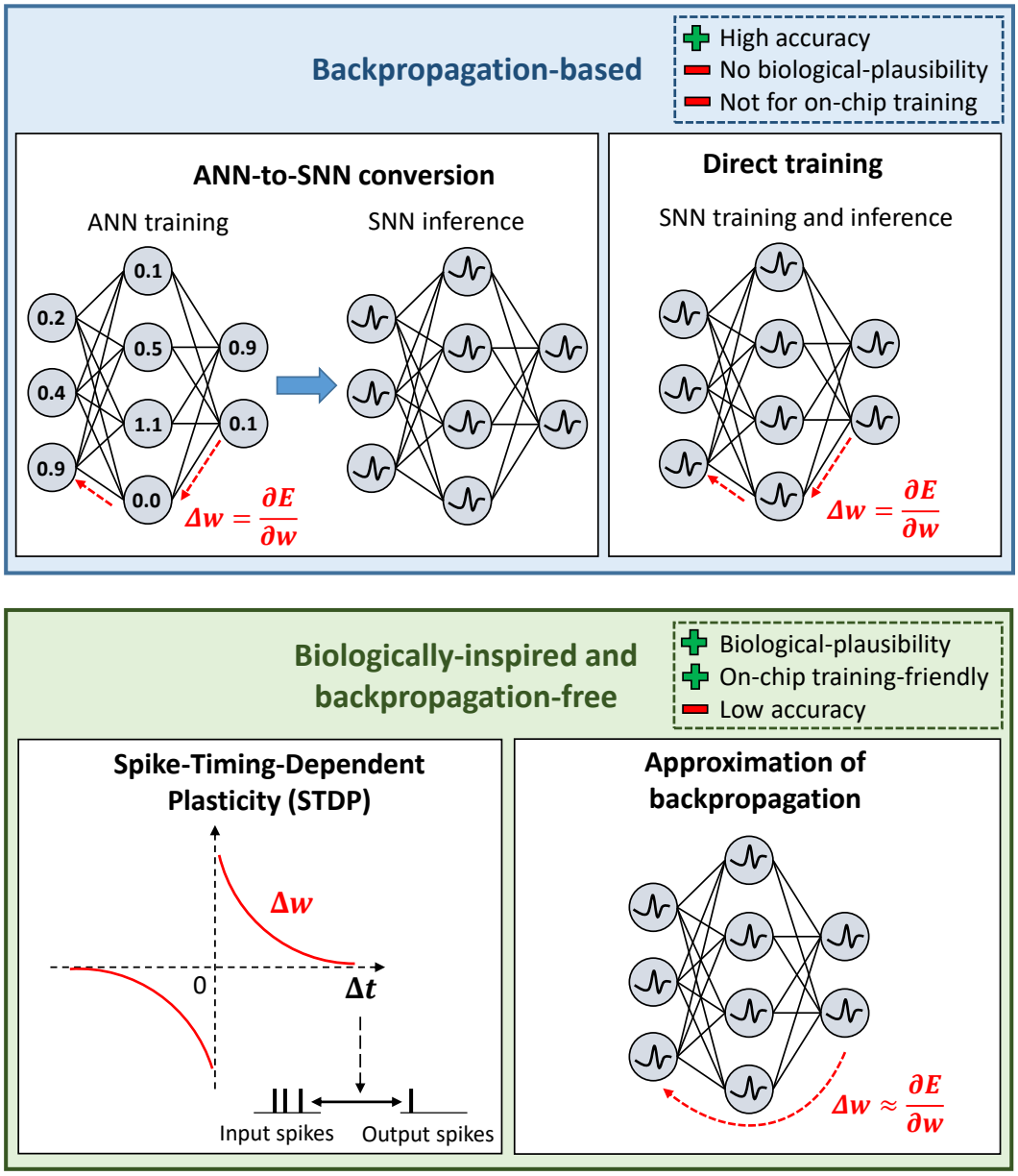


FIGURE 2.4: **Training strategies for Spiking Neural Networks.** Backpropagation-based strategies (top) are based on gradient descent, in which a cost function **E** is minimized and weight updates δw are performed by computing the derivative of **E** with respect to the weight **w**. Training with gradient descent is performed either on a equivalent ANN and then its weights are transferred to the SNN (ANN-to-SNN conversion), or directly on the SNN. Biologically-inspired and backpropagation-free strategies (bottom) are based either on the biologically-plausible unsupervised Spike-Timing-Dependent Plasiticity rule, or on approximations of the backpropagation algorithm.

with 98.4% accuracy. Lee et al., 2018 use STDP as a pre-training, followed by a fine-tuning with backpropagation and showed that the STDP pre-training gives a better accuracy (99.28% on MNIST) than a random initialization. Mozafari et al., 2019b add reinforcement learning to create a reward-modulated STDP, achieving 97.2% on MNIST without requiring an external classifier. Srinivasan et al., 2019 introduced a Stochastic-STDP learning rule for a SNN with binary weights. With a residual CNN composed of four layers, they achieved 66.23% on the CIFAR-10 dataset. However, the last fully connected layer of their network is trained with the standard backpropagation algorithm. Moreover, the convolutional layers of the network are connected to each other (using a direct path and residual connections) and to the fully connected layer, making the fully connected layer critical for the classification tasks. This shows that the STDP cannot make full use of the depth of the network. Indeed, the accuracy of their model increases with the number of convolutional layers up to three, after what the accuracy starts to deteriorate. Without adding supervision to the training process, Thiele et al., 2018 propose STDP with two Integrate and Fire units per neuron to decouple the learning and inference with SDTP, enabling the training of all layers simultaneously in an event-driven fashion. Therefore, this system is very suitable for online learning, where the learning is performed at the same time as inference, and achieves 96.58% accuracy on MNIST. Later, Fu et al., 2021 remarkably demonstrate 93% accuracy on CIFAR-10 with an Ensemble SNN trained with STDP. The Ensemble SNN are composed of identical CNNs (with three convolutional layers) and a voting layer is used at the end to determine the classification output. Finally, STDP can also be used in recurrent topologies. Chakraborty et al., 2023 demonstrate that training with STDP in recurrent SNNs can be significantly improved by adding heterogeneity in the neurons (varying firing and relaxation dynamics) and in the STDP itself (varying learning dynamics for each synapse). They achieve 96.54% accuracy on DVSGesture with a recurrent SNN with 2000 hidden neurons, while the SNN trained with backpropagation reaches 98.12%.

The STDP rule makes it possible to train SNNs in an unsupervised and biologically-plausible manner and is suitable for on-chip learning due to its locality. However, although Tavanaei et al., 2019; Eshraghian et al., 2022 have shown that STDP can approximate backpropagation update rules, it is not the most appropriate rule to train deep networks with high accuracy in the context of deep learning applications.

Approximations of Backpropagation

Training approaches based on approximations of backpropagation use supervised training with gradient descent, but do not apply the standard layer-by-layer backpropagation, either to increase the biological plausibility or to reduce the training cost. Indeed, the layer-by-layer backpropagation algorithm is difficult to implement efficiently in neuromorphic hardware, and is also considered highly implausible in the brain, mainly for the non-locality (in both space and time) of the computations. Moreover, it requires precise calculation of real-valued gradients, a separation and synchronization of the forward and backward pass while storing all the activations, and symmetry of weights in both directions (Bengio et al., 2016).

In ANNs, "random backpropagation" approaches, first introduced in Lillicrap et al., 2016, are proposed to relax some constraints of the backpropagation phase, at the cost of a degradation in accuracy. Some works have applied these methods to SNNs.

For instance, Zenke et al., 2018 use Feedback Alignement (Lillicrap et al., 2016), which removes the constraint of symmetric weights, to SNNs. Kaiser et al., 2020b apply Sign-concordant Symmetry (Liao et al., 2016), which uses random magnitudes for feedback weights but preserve the signs. Neftci et al., 2017; Lee et al., 2020c; Bellec et al., 2020; Kaiser et al., 2020a use Direct Feedback Alignment (Nøkland, 2016) to SNNs. In this variant, the output error is directly fed to each hidden layer through fixed random connectivity matrices, removing the layer-wise locking constraint (in addition to the symmetric weights) during the backward pass. Neftci et al., 2017 propose an event-driven Direct Feedback Alignment with dual compartments neurons (one for the forward and one for the backward pass). Due to dual neurons, the model needs only two additions and one comparison to perform the synaptic weight update, provided auxiliary neurons to store the accumulated error. Bellec et al., 2020 trained a recurrent SNN with adaptive neurons with a form of Direct Feedback Alignment. The training algorithm, called e-prop, is an approximation of the Backpropagation Through Time (BPTT, which is used to train RNNs), using eligibility traces recording the pre- and post-synaptic activity and an error signal propagated through direct feedback connections. A recurrent SNN with adaptive neurons trained with e-prop achieves 26.4% error rate for speech recognition on the TIMIT dataset (Garofolo et al., 1993), compared to 24.7% error when trained with BPTT. In addition, Bohnstingl et al., 2022 implement e-prop on neuromorphic hardware leveraging analog synapses and IMC, thus enabling efficient online and real-time learning. Moreover, Mostafa et al., 2018 propose a training method based on local errors that removes the layer-wise locking constraint also during the forward pass, making the learning of each layer independent. The errors are generated locally in each layer using fixed random (sign-concordant) auxiliary classifiers and each layer try to minimize its local error. Kaiser et al., 2020b apply this method to SNNs showing 95.54% accuracy on DVSGesture. Moreover, the gradients can be computed locally at each timestep, enabling online learning. Indeed, Guo et al., 2022 implemented local learning on a neuromorphic chip demonstrating 4x and 10x higher efficiency compared to a STDP method and a Direct Feedback Alignement method, respectively, in terms of trade-off between energy, speed, resources and accuracy. Note that some of these spiking algorithms (such as Neftci et al., 2017; Zenke et al., 2018; Bellec et al., 2020; Kaiser et al., 2020b) are sometimes called "three-factor" learning rules to enhance their proximity with biological learning rules. "Three-factor' stands for two factors corresponding to the pre and postsynaptic activity (reminiscent of the STDP learning rule) and a third factor corresponding to an error signal (similarly to reinforcement leaning).

In conclusion, training SNNs based on approximations of the backpropagation algorithm can be relevant to target efficient on-chip training or to explore biologically-plausible learning. However, for now, these approximations decrease the accuracy compared to standard backpropagation.

2.2.2 Backpropagation-Based Training

Backpropagation-based training methods are able to accurately train deep networks. However, applying backpropagation to SNNs is challenging due to the nature of the spiking activation function, which has derivative equals to zero everywhere except in zero where it is infinite (see Fig. 2.5). Different strategies have been proposed to mitigate this problem, such as approximating the derivative with a surrogate gradient

(Neftci et al., 2019), or directly differentiating the spike times (Mostafa, 2018). An alternative solution is to convert a trained ANN to a SNN formalism, also called indirect training or ANN-to-SNN conversion, which bypasses the training difficulty of SNNs.

ANN-to-SNN Conversion

The ANN-to-SNN conversion is an indirect training strategy consisting in training an ANN and then mapping the trained weights to a SNN, assuming equivalence of the SNN computing units to the ANN ones. The ANN is trained under constraints to better fit the SNN model. Then, either the thresholds of the spiking neurons or the weights are normalized equivalently, so that the transfer function (input-output mapping) of the SNN unit matches the transfer function of the ANN unit (Diehl et al., 2015; Sengupta et al., 2019).

Conversion with rate coding. Rate coding is a straightforward approach to conversion, in the case of using IF SNN neurons and Rectified Linear Unit (*ReLU*) activation functions for ANNs (Cao et al., 2015; Rueckauer et al., 2017). Indeed, the firing rate of an IF neuron approximates the analog output of a *ReLU* function (linear on the positive x-axis and zero otherwise). However, the conversion process results in errors in some cases, for instance when the ANN activation is too high and cannot be accurately represented by the spike rate given a fixed simulation duration. An effective data-based weight normalization, consisting in rescaling the weights in each layer according to the maximum ANN activation in the corresponding layer within the training set, is presented in Diehl et al., 2015 to mitigate this problem. Another solution is proposed in Sengupta et al., 2019, which balances the thresholds in each layer according to the maximum SNN activation, instead of ANN activation. They report high accuracy for a converted SNN with VGG-16 network architecture, such as 91.55% on CIFAR-10 and 69.96% on ImageNet using 2500 inference timesteps. The accuracy can be further improved using a soft reset mechanism instead of a hard reset, as proposed in Han et al., 2020b. The soft reset consists in subtracting the threshold value from the membrane potential after the neuron fires a spike, instead of setting it to the reset potential value. The residual membrane potential above the threshold is thus kept for the next spike, which reduces the loss in the spiking quantization process. Their method yields a near loss-less conversion, showing 93.63% accuracy on CIFAR-10 and 73.09% on ImageNet with a VGG-16 architecture using 2048 and 4096 timesteps, respectively. Similar results are shown with ResNet-20 and ResNet-34 on CIFAR-10 and ImageNet achieving 91.36% and 69.89% accuracy, respectively. However, these methods require hundreds to thousands of inference timesteps, leading to a very high latency and a degraded energy efficiency, according to Roy et al., 2019. The long inference time required to achieve high accuracy is inherent to the equivalence chosen for the conversion. Indeed, the *ReLU* activation function approximates the firing rate of the IF model only if the SNN inference is discretized with a sufficient number of timesteps. More recently, Li et al., 2021; Li et al., 2022; Bu et al., 2022a leverage quantization theory to further reduce the conversion loss between ANN and SNN under low latency constraints. Indeed, a spiking neuron can be seen as quantization function, with the number of timesteps defining the quantization precision. For instance, Bu et al., 2022b propose to replace the standard *ReLU* in the ANN model by the quantization clip-floor-shift activation function, which better approximates the firing rate of the IF model. Indeed, by clipping the

ReLU, they suppress conversion errors due to large ANN activations. Moreover, by quantizing the ANN activation (flooring), they remove conversion errors due the discretization of the SNN activation, if the quantization step is properly chosen according to the time discretization of the SNN inference. Indeed, by clipping (resp. flooring), they suppress the conversion errors due to large activations in the ANN (resp. due to the discretization of the SNN activation), if the quantization step is properly chosen according to the time discretization of the SNN inference. Therefore, they obtain SNNs with high accuracy using a much smaller number of timesteps. In addition, they propose an optimal initialization of the membrane potentials to decrease the SNN latency. Indeed, they show that initializing membrane potentials to zero makes the SNN start to fire late, especially in the deepest layers, while initializing it to half of the spiking threshold considerably decreases the time to first spike of the neurons, resulting in higher accuracy at lower latency. For instance, on ImageNet with ResNet-34, they achieve 69.37% (resp. 72.35%) accuracy with 32 (resp. 64) timesteps. However, there is still a gap between the accuracy of the ANN and the converted SNN due to unevenness errors, resulting from the fact that a different order of input spikes produces a different output (Bu et al., 2022b), which is inherent to the conversion process. Li et al., 2022 propose to mitigate the unevenness errors (which they call "occasional noise") by allowing the emission of negative spikes to correct a potential surplus of fired spikes. Notably, they achieve 72.91% (resp. 74.36%) accuracy with ResNet-50 on ImageNet using only 5 (resp. 10) timesteps. However, this requires relaxing the constraint of binary spikes, by allowing both positive and negative spikes.

Conversion with temporal coding. Another approach to conversion is based on temporal coding. This approach is attractive because the number of spikes emitted can potentially be decreased drastically, thus further reducing the energy consumption. This is first proposed in Rueckauer et al., 2018 using the equivalence between the activation value of the ANN unit and the inverse of the spike time of the SNN unit. In Han et al., 2020a an accuracy as high as that obtained with rate coding in Han et al., 2020b is demonstrated on CIFAR-10 and Imagenet using VGG-16 and ResNet architectures, with at most two spikes per neuron. They propose a novel temporal coding with one positive and one negative spikes per neuron. In addition, they introduce a threshold balancing method to improve the accuracy while using fewer timesteps compared to the rate-based conversion of Han et al., 2020b. Furthermore, Stöckl et al., 2021 propose a temporal code associated with a new spiking neuron model using $logN$ different values of spike times to transmit integers between 1 and N in order to reduce the required temporal resolution. They achieve 83.57% accuracy on ImageNet with the EfficientNet-B7 architecture (75.10% with ResNet-50), with on average less than 2 spikes per neuron per inference. However, the proposed neuron model is complex, requiring additional parameters and additional state functions computed at each timestep, and might be costly to implement on neuromorphic hardware. More recently, Stanojevic et al., 2022 show an exact mapping between *ReLU* neurons and SNN neurons with a linear latency coding, allowing to reach a loss-less ANN-to-SNN conversion. However, the required temporal resolution to simulate this loss-less conversion is not indicated.

To conclude, the ANN-to-SNN conversion results in high accuracy. However, there is still a gap between the accuracy of the ANN and the converted SNN due the difficulty of having a loss-less conversion when aiming at low inference latency. In addition, the conversion process does not allow the optimization of the temporal dynamics

of the SNN, contrary to direct training approaches (Rathi et al., 2021a).

Direct Training With Backpropagation

Direct training with backpropagation is another training strategy for accurate SNNs. In order to directly apply backpropagation to SNNs, different strategies are used to circumvent the problem of the non-differentiability of the spiking activation function, such as approximating the derivative with a surrogate gradient (Neftci et al., 2019), or directly differentiating the spike times as proposed in Mostafa, 2018. The surrogate gradient technique consists in approximating the neuron's activation function (a step function) by a differentiable function during the backward pass, to enable informative gradients to backpropagate through the layers (see Fig. 2.5). The approach was first introduced by Hinton, 2012; Bengio et al., 2013 with the straight-through estimator used to train quantized neural networks. Furthermore, the SNN inference being discretized in timesteps, there are several strategies to apply backpropagation to SNNs (see Fig. 2.6). We classify these methods into three categories. (1) The spatial approach uses accumulated quantities which are retrieved at the end of the inference phase (such as the spike count or membrane potential value) to serve as an activation value for each neuron. Then, backpropagation can be applied on these quantities. This strategy does not take into account the SNN temporal dynamics (i.e. the precise timing and order of the spikes and temporal components of the SNN model). (2) The single-spike approach uses only one spike per neuron (using a latency temporal coding), and computes directly the spike time of each neuron during the inference phase. Then, the backpropagation algorithm can be applied using the spike time of each neuron as its activation. (3) The spatio-temporal approach considers the SNN as a RNN and performs the backpropagation on each timestep using BPTT. Hence, this method considers both temporal and spatial dynamics of SNN.

Spatial approaches, such as Lee et al., 2016; Thiele et al., 2019; Wu et al., 2019a, use a rate-coding strategy with IF neurons and instantaneous synapses and do not consider temporal dependencies in the gradient computation. They consist in approximating the SNN forward pass during the training in order to obtain a lighter backpropagation, only in the spatial domain, as in ANN training. For instance, Lee et al., 2016 consider the membrane potential without the spiking discontinuities, using low-pass filtered spike signals. Therefore the signal is continuous (considering the spiking activity as noise) and backpropagation can be applied on it. They achieve 99.31% accuracy on MNIST with a 4-layer CNN architecture. A different method is proposed in Wu et al., 2019a. They define the equivalent of the neuron activation as the sum of its spikes produced during the simulation time, which can be used for backpropagation. They achieve 99.26% accuracy on MNIST with a 3-layer CNN architecture. However, they argue that considering only the spike count generates a quantization error, as the surplus membrane potential of spiking neurons is not taken into account, which could become a problem for deeper neural networks. Furthermore, Thiele et al., 2019 demonstrate that the backpropagation phase can also be realized with spikes by considering the error in a discrete form. Therefore, the same hardware infrastructure can be used for both inference and learning, which makes it attractive for on-chip learning. Moreover, they show that the spike discretization error can be reduced to zero by adding some constraints on the ANN. Therefore, they can perform the offline training directly

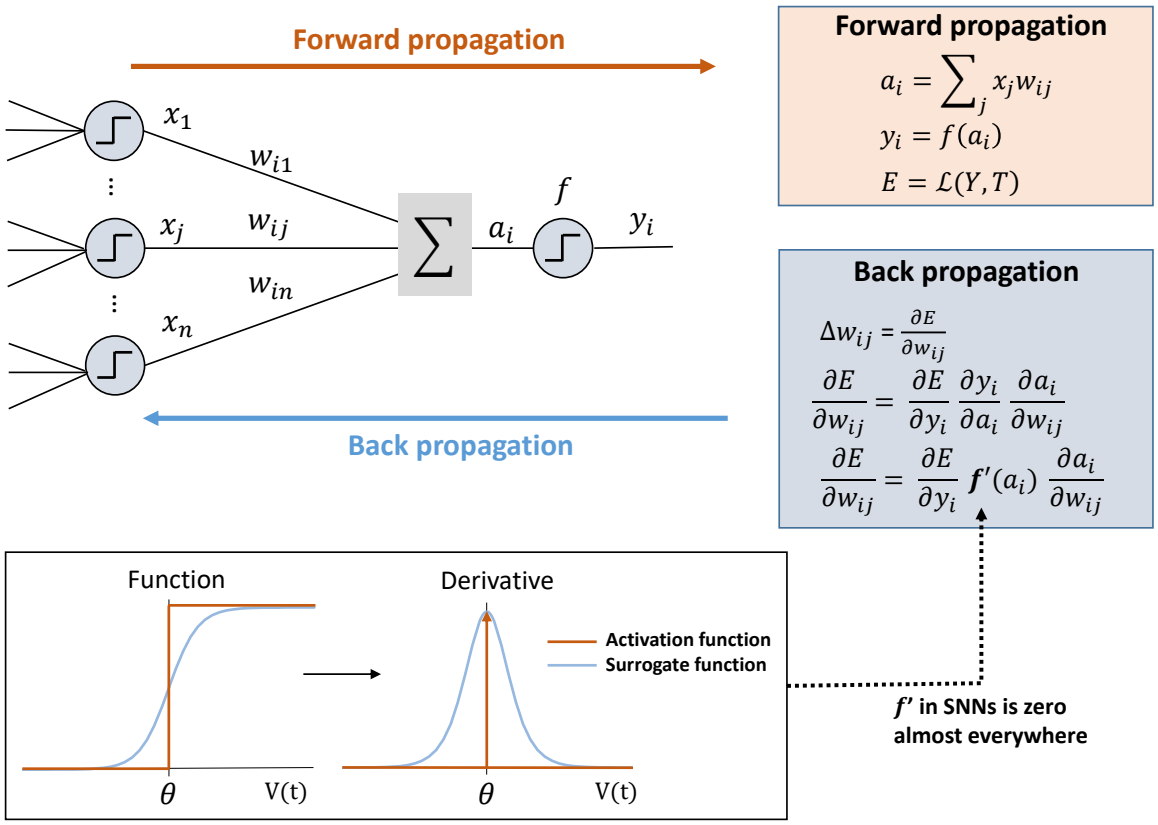


FIGURE 2.5: **Supervised learning with backpropagation in SNNs.** In the forward pass, the inputs are propagated through the layers resulting in output **Y**. During the backward pass, **Y** is compared to a target **T** using a loss function **L**, defining the cost **E** to be minimized with gradient descent. Then each weight **w** is updated according to the derivative of the cost **E** with respect to **w**. This derivative is computed using the chain rule, which requires to compute the derivative of all previous operations. This allows to backpropagate the errors through all the layers of the network. However, in SNNs, the activation function **f** of neurons has the derivative equals to zero everywhere except in θ where it is infinite. Therefore, the derivative of a surrogate function is used to compute the gradients during the backward pass.

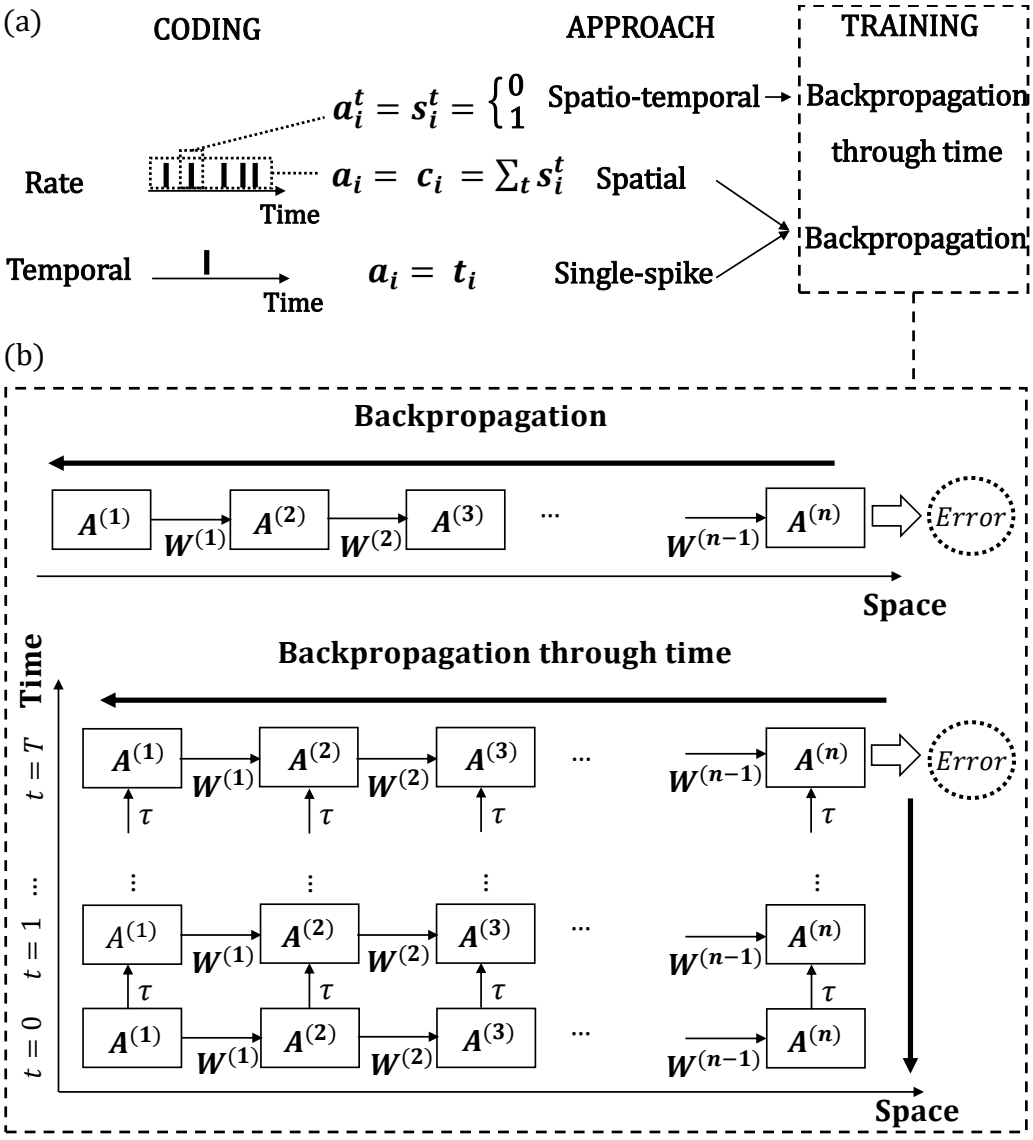


FIGURE 2.6: (a) Backpropagation-based learning algorithms. Spatial and spatio-temporal approaches use a rate coding while the single-spike approach use a temporal (latency) coding. On the one hand, the spatio-temporal approach considers the activation of each neuron at each timestep a_i^t, corresponding to the emission or not of a spike s_i^t. BPTT is used to backpropagate the error in both space and time dimensions. On the other hand, the spatial and single-spike approaches consider for each neuron a single activation a_i for the forward pass, which can correspond to the spike count c_i for the former or the timing of the unique spike emitted by the neuron t_i for the latter. Therefore, the backpropagation is used to backpropagate the error only in the space dimension. (b) Backpropagation and BPTT training. Notations: n number of layers, T number of timesteps used in the SNN inference, $A^{(l)}$ activation of neurons in layer l, $W^{(l)}$ weight vector from layer l to $l+1$, τ membrane potential and postsynaptic potential update (for LIF neurons and continuous synapses).

with the equivalent ANN. They achieve 89.99% accuracy on CIFAR-10 with a 8-layers VGG architecture. However, because they perform training and inference with the equivalent ANN, they do not indicate the number of inference timesteps that would be used for the inference with the SNN. Note that the spatial approach resembles the ANN-to-SNN conversion with rate coding, however the training targets are different. In conversion methods, the ANN is trained under constraints and then, there is a conversion procedure to transfer the ANN trained weights to the SNN. On the other hand, in the spatial approach, the SNN is directly trained, but viewed as an ANN, and thus can be trained in a similar way using accumulated quantities during the SNN forward pass. Hence, contrary to the ANN-to-SNN conversion, there is only one network and no conversion procedure. However, the most recent ANN-to-SNN conversion techniques resemble the spatial SNN training. Indeed, they convert a SNN from an ANN that matches more closely the SNN such that the differences between the two models decrease and the conversion becomes more straightforward (Bu et al., 2022b).

Single-spike approaches, such as Mostafa, 2018; Comşa et al., 2022; Göltz et al., 2020; Zhang et al., 2020; Sakemi et al., 2023; Kheradpisheh et al., 2020; Zhou et al., 2021, apply backpropagation in SNNs by directly differentiating the spike times. These methods have been applied to image processing, the spatial information contained in the images being directly converted to the spike timing using a temporal (latency) coding. These approaches have the advantage of using at most one spike per neuron and thus appear promising for energy-efficient hardware implementations. The single-spike approach was introduced very early with the SpikeProp learning rule from Bohte et al., 2002. SpikeProp defines the firing time of neurons as a function of their membrane potential and thus approximates their derivative using the changes of the membrane potential around the firing time. Later, Mostafa, 2018 demonstrated that by using single-spike IF neurons with exponential synapses, the differential equation of the neuron membrane potential has a simple solution. Due to this analytic input-output relation, the spike times can be computed directly without simulating the inference with timesteps. Therefore, there is no need to use BPTT, but a direct backpropagation only on the spike times is possible. They achieve 97.2% accuracy on MNIST with a Fully-Connected (FC) network with 800 hidden neurons. Comşa et al., 2022 and Göltz et al., 2020 take their inspiration from Mostafa, 2018 but use IF neurons and synapses with alpha synaptic kernel, and LIF neurons and synapses with dual exponential kernel, respectively. Their models are more biologically plausible, but the differential equations have complex solutions. Performance in hardware is demonstrated in Göltz et al., 2020 by implementing the algorithm in BrainScaleS-2 (Pehle et al., 2022). They yield 95.9% accuracy with $25\mu J$ per classification on MNIST (16x16 images) with a FC network with 128 hidden neurons. Zhang et al., 2020 uses IF neurons with linear synapses, taking inspiration from the *ReLU* units used in ANNs, demonstrating 99.2% accuracy on MNIST with a 5-layer CNN architecture. Besides, Kheradpisheh et al., 2020 use instantaneous instead of continuous synapses with IF neurons. However, they must approximate the derivative of the spike time with regards to the membrane potential, and hence the gradients are not exact. They obtain 97.4% accuracy on MNIST with a FC network with 400 neurons in the hidden layer. Zhou et al., 2021 extend the work of Mostafa, 2018 by proposing an efficient way of computing the spike times taking advantage of parallel tensor computations in deep learning frameworks running on GPUs to speed-up the offline training. They achieve 92.68% and 68.8% accuracy on CIFAR-10 and

ImageNet using VGG-16 and GoogleNet architectures, respectively. However, as they directly compute the spike times without simulating the SNN dynamics, the number of timesteps that would be used for the SNN inference with the desired temporal resolution in hardware is unknown. In addition, single-spike approaches are hardly compatible with dynamically changing input data (such as spatio-temporal data), as neurons can fire only once, according to Eshraghian et al., 2022.

Spatio-temporal approaches, such as Wu et al., 2018; Shrestha et al., 2018; Zenke et al., 2018; Jin et al., 2018; Wu et al., 2019b; Lee et al., 2020a; Zhang et al., 2020; Ledinauskas et al., 2020; Kim et al., 2020; Fang et al., 2020a; Zheng et al., 2021; Fang et al., 2021b; Fang et al., 2021a; Deng et al., 2022; Duan et al., 2022, use a rate-coding strategy, but propagate the gradient both in spatial and temporal dimensions using BPTT. The majority of works use LIF neurons with instantaneous synapses and rate-coded loss functions (based on the output firing rate, or output membrane potential in the case of non-spiking neurons). Wu et al., 2018 introduce a spatio-temporal backpropagation for SNN based on an iterative LIF model and approximate the non-differentiable spiking activity with a surrogate gradient. They demonstrate 99.42% accuracy on MNIST with a 4-layer CNN architecture. Lee et al., 2020a follow the approach of Lee et al., 2016 by considering the membrane potential without the spiking discontinuities, but with a leak in the neuron model. They achieve 90.95% accuracy on CIFAR-10 with a ResNet-11 using 100 timesteps for inference. However, using a surrogate gradient has the effect of smoothing the spiking activity and leads to a degraded accuracy of the computed gradients according to Jin et al., 2018; Zhang et al., 2020. Therefore, Jin et al., 2018; Zhang et al., 2020 propose an alternative backpropagation method at the spike train level. For instance, Zhang et al., 2020 decompose the derivative of the spiking activation with regards to the membrane potential in two factors, one accounting for the inter-neuron dependencies and one accounting for the intra-neuron dependencies. However, in their neuron model, post-synaptic potentials are transmitted between layers instead of spikes, and hence they loose the advantage of a spiking implementation (with sparse and binary activations). Note that the computational and memory cost of training with BPTT is important, as this method requires storing the activations and computing the gradient at all timesteps. For instance, training VGG-16 on CIFAR-10 for one epoch of BPTT using 100 timesteps takes 78 min and 9.36 GB of GPU memory (using Nvidia GeForce RTX 2080 Ti TU102 GPU with 11GB memory) according to Rathi et al., 2020. For comparison, the VGG-16 ANN training of one epoch requires only 0.57 min and 1.47 GB, which is x137 less time and x6 less memory. Moreover, Ledinauskas et al., 2020 argued that, for SNNs trained with BPTT, reducing the number of timesteps is crucial, not only to reduce the latency and energy consumption, but also to improve the training convergence. Indeed, similar to RNNs, the vanishing and exploding gradients problem can appear. Note that, although most of the cited papers consider applications with static data (such as images), training with BPTT is particularly useful in the case of spatio-temporal data. In this case, BPTT allows to optimize the SNN dynamics taking into account the temporal dynamic of the input data (while there is no temporal dynamics in rate-coded static data).

TABLE 2.1: Comparison of backpropagation-based direct training strategies on static vision datasets

BP training	Paper	Topo.	Strategies	Timesteps	Acc. (%)
CIFAR-10					
Spatial	Thiele et al., 2019	VGG-8	/	/	89.99
Spatio-temporal	Wu et al., 2019b	VGG-8	NN, DO, ENC, VOT	12	90.53
	Lee et al., 2020a	ResNet-11	DO	100	90.95
	Zhang et al., 2020	VGG-8	ENC	5	91.41
	Ledinauskas et al., 2020	ResNet-11	BN, DO, SG tuning	20	90.20
	Kim et al., 2020	VGG-9	BN	25	90.50
	Zheng et al., 2021	ResNet-19	BN, ENC, VOT	6	93.16
	Fang et al., 2021b	VGG-8	BN, DO, ENC, VOT	8	93.50
	Deng et al., 2022	ResNet-19	BN, ENC	6	94.50
	Duan et al., 2022	ResNet-19	BN, ENC, VOT	6	94.71
	Castagnetti et al., 2023	ResNet-18	ENC, v_{th} tuning	4	94.65
Single-spike	Zhou et al., 2021	VGG-16	/	/	92.68
ImageNet					
Spatio-temporal	Zheng et al., 2021	ResNet-34	BN, ENC, VOT	6	67.05
	Fang et al., 2021a	ResNet-152	BN, ENC	4	69.26
	Deng et al., 2022	ResNet-34	BN, ENC	4	68.00
	Duan et al., 2022	ResNet-34	BN, ENC, VOT	4	68.28
Single-spike	Zhou et al., 2021	GoogLeNet	/	/	68.8

NN: neuron normalization, BN: batch normalization, DO: dropout, ENC: encoding layer, VOT: voting layer, SG: surrogate gradient, v_{th}: neuronal threshold.

2.3 Improving Accuracy and Efficiency of Spiking Neural Networks

2.3.1 Improving Backpropagation-Based Training

Backpropagation-based training allows to achieve better accuracy than biologically-inspired learning rules. However, the accuracy of SNNs is still significantly lower than that of ANNs and the latency is important. Therefore, additional strategies are required to improve the supervised training of SNNs with backpropagation in terms of accuracy, latency and spike sparsity. The state-of-the-art on direct training with backpropagation-based methods (spatial, spatio-temporal and single-spike) on static and neuromorphic vision datasets is summarized in Tables 2.1 and 2.2, while showing the effect of using the improvements described in the following subsections.

Adapting ANN Techniques to SNNs

SNNs can benefit from techniques developed to improve ANN training, such as regularization, normalization, optimized topologies and quantization theory. For instance, dropout (Srivastava et al., 2014) is an effective regularization technique consisting in randomly disconnecting some units of a layer during the training to avoid the network relying too much on certain connections. This technique is transferable to SNNs

TABLE 2.2: Comparison of backpropagation-based algorithms on neuromorphic vision datasets

BP training	Paper	Topo.	Strategies	Timesteps	Acc. (%)
DVSGesture					
Spatio-temporal	Fang et al., 2020a	5-layer CNN	synaptic kernel optimization	/	96.09
	Fang et al., 2021b	VGG-7	BN, DO, ENC, VOT	20	97.57
	Zheng et al., 2021	ResNet-17	BN, ENC, VOT	40	96.87
CIFAR-10-DVS					
Spatio-temporal	Wu et al., 2019b	VGG-5	NN, DO, ENC, VOT	20	60.5
	Kim et al., 2020	VGG-7	BN	20	63.2
	Zheng et al., 2021	ResNet-19	BN, ENC, VOT	10	67.8
	Fang et al., 2021b	VGG-6	BN, DO, ENC, VOT	20	74.8
	Deng et al., 2022	VGG-9	BN, ENC	10	83.17
	Duan et al., 2022	VGG-9	BN, ENC, VOT	10	84.90

NN: neuron normalization, BN: batch normalization, DO: dropout, ENC: encoding layer, VOT: voting layer.

and was used in many works such as Wu et al., 2019b; Rathi et al., 2020; Lee et al., 2020a; Ledinauskas et al., 2020; Rathi et al., 2021b; Fang et al., 2021b. Batch Normalization (BN) (Ioffe et al., 2015) is a powerful normalization technique widely used to train deep ANNs. It consists in rescaling the activations of a layer, and learning this scaling per batch, in order to maintain the variance of the activations throughout the network, which leads to better convergence. Several works propose to transfer the BN technique to SNN training, such as Ledinauskas et al., 2020; Kim et al., 2020; Zheng et al., 2021; Duan et al., 2022. Kim et al., 2020 show that standard BN should not be applied directly to SNNs, because it considers the timesteps all at once. Instead, they propose a BN "through time" (BNTT) to decouple the parameters of the BN across the timesteps. They report a decreased number of spikes per inference by one order of magnitude compared to the BPTT without BN. They show x9 efficiency gain compared to the ANN version in terms of AC and MAC operations using the energy values of Horowitz, 2014. Zheng et al., 2021 propose a threshold-dependent spatio-temporal BN (tdBN) that normalizes the variance of the inputs to the threshold (of the spiking activation function). They demonstrate scalability to deep residual networks with high accuracy while using fewer timesteps. In addition, they report sparse spiking activity (less than 2 spikes per neuron per inference on average). Duan et al., 2022 improve over tdBN and BNTT by proposing a temporal efficient BN (TEBN). Compared to BNTT and similarly to tdBN, a unique set of BN parameters (weight and shift) and a unique set of statistics are used for all timesteps. However, contrary to tdBN, the presynaptic inputs are rescaled with a different weight for each timestep. This allows to keep the temporal coherence of information between timesteps while reducing the computational complexity compared to BNTT. Alternatively, a neuron normalization method especially designed for SNN is proposed in Wu et al., 2019b, based on auxiliary neurons at each layer to help balance the input currents (preactivations).

Scalability of SNNs can also be improved using optimized network architectures. For instance, the ResNet architecture can alleviate the gradient vanishing problem by adding residual shortcut connections, which enables the effective training of deeper networks He et al., 2016. Lee et al., 2020a; Ledinauskas et al., 2020; Zheng et al., 2021;

Fang et al., 2021a achieve high accuracy using ResNet architectures and regularization techniques. For instance, Ledinauskas et al., 2020 and Zheng et al., 2021 show scalibility up to a 50-layer ResNet on CIFAR-100 and ImageNet, respectively. However, the accuracy gap with respect to ANN for the same architecture is still high. On ImageNet, the SNN ResNet-50 in Zheng et al., 2021 yields 64.88% accuracy while the ANN ResNet-50 achieves 76.13% (*Pytorch torchvision models* 2021), both being trained with BN. Notably, Fang et al., 2021a propose a novel implementation of the identity mapping in spiking ResNet, which can mitigate the problem of SNNs scaling with depth. Indeed, they are able to scale to very deep residual networks while increasing the accuracy with depth. For instance, they reach 69.26% accuracy on ImageNet with ResNet-152 using only 4 timesteps.

Leveraging quantization theory developed for quantized ANN training can further allow to improve the SNN accuracy. This idea, which has been explored to reduce the conversion loss in ANN-to-SNN conversion methods with low-latency constraints (Li et al., 2022; Bu et al., 2022a), has been later used in direct training (Castagnetti et al., 2023). Castagnetti et al., 2023 view the spiking activation function as a quantization function, where the timesteps are seen as quantization intervals. They propose to directly target the minimization of the error of this quantization function by training layer-wise SNN thresholds. In particular, using time-varying thresholds (i.e. with a different value at each timesteps) allows to further decrease the quantization loss, in analogy with non-uniform quantization. Notably, they achieve 71.42% accuracy on CIFAR-100 with ResNet-18 using only 4 timesteps.

Improving Encoding and Decoding

Input encoding and output decoding of the network seriously impact the SNN accuracy and latency according to Rueckauer et al., 2017; Wu et al., 2019b; Garg et al., 2020; Deng et al., 2020.

Encoding. The higher the number of timesteps used in the inference phase, the higher the precision of the encoding and in turn, the higher the network accuracy. However, this induces an increased inference latency. Therefore, reducing the number of timesteps while preserving the accuracy is challenging. In the case of real-valued signals (non-spiking data), an efficient solution is to use an encoding layer, also called direct input encoding, as in Rueckauer et al., 2017; Wu et al., 2019b; Deng et al., 2020; Rathi et al., 2021b; Wu et al., 2021; Fang et al., 2021b; Zheng et al., 2021. It consists in directly feeding the real values to the first layer at each timestep, without discretization of the input, the discretization process with the spikes being done in the first layer. Such encoding layer is thus a hybrid ANN-SNN layer, as the synapses perform real-valued input-weight multiplications but the neurons are spiking units (see Fig. 2.7). Deng et al., 2020 report that by using the encoding layer, a VGG-5 trained with BPTT on CIFAR-10 could achieve 74.23% accuracy instead of 63.19% with a rate-based encoding. Using an encoding layer can also reduce the latency for the same accuracy. For instance with the encoding layer, the same accuracy is achieved as rate-based encoding with only 3 timesteps instead of 15. Similar conclusions are derived in Kim et al., 2022b. Note that this hybrid layer (introduction multiply operations) must be supported in the neuromorphic hardware used for inference, or computed at the interface between the data and neuromorphic hardware.

Decoding. A simple way to increase the precision of the output layer is to apply the loss function on the high precision membrane potential of the output neurons instead of their spike rate, as shown in Lee et al., 2020a; Rathi et al., 2020; Rathi et al., 2021b. Alternatively, population decoding can be used to improve the robustness of the classification when using the output spike rates as in Wu et al., 2019b; Fang et al., 2021b; Zheng et al., 2021; Duan et al., 2022. In this decoding scheme, each group of output neurons represents a class, and the choice is made based on a voting strategy. Besides, Deng et al., 2022 show that by slightly modifying the loss function, they could achieve better convergence of the SNN trained with a spatio-temporal approach. Indeed, instead of computing the cross-entropy loss function on the average of the SNN outputs, they compute the average of the loss at each timestep. They achieve especially good results with spatio-temporal data such as neuromorphic datasets (83.17% on DVS-CIFAR10).

Wide Network Architectures

Using wider network architectures, by increasing the number of neurons per layer, improves the accuracy, especially when the number of timesteps is low. For instance, Lee et al., 2020a; Rathi et al., 2020; Zheng et al., 2021; Rathi et al., 2021b; Ledinauskas et al., 2020 achieving low latency with high accuracy on CIFAR-10 use very large ResNet architectures (11 to 18M parameters) compared to those usually used for the CIFAR-10 task (such as ResNet-20 with only 0.27M parameters, but yet the ANN version achieves 91.25% accuracy). On ImageNet, Zheng et al., 2021 report that by doubling the number of filters per convolutional layers in the ResNet-34 architecture, they increase the accuracy from 63.72% to 67.05%. Interestingly, when they compare the SNN with the ResNet-50 and ResNet-34 original architectures, the ResNet-50 yields better accuracy (64.88%), but not compared to the large ResNet-34. This shows that, in SNNs, from a certain depth, increasing the width of the network is more beneficial compared to further increasing the depth. Indeed, the number of neurons is increased, which can have a similar effect to increasing the precision of a smaller number of neurons. This is on par with Mishra et al., 2017 demonstrating that wider architectures can improve the accuracy of reduced-precision ANNs (with quantized weights and activations). Indeed, SNNs with few timesteps behave similarly to ANNs with highly-quantized activations.

Training Hybridization

Hybrid training approaches have also been proposed to further reduce the cost of the off-chip training or the hardware inference, while increasing the accuracy.

ANN-SNN network hybridization. Mixing ANN and SNN layers (see Fig. 2.7) is one strategy to improve the accuracy. For instance, Panda et al., 2020 use a network with ANN layers at the inputs to improve the encoding accuracy, and SNN layers at the output, the whole network being trained with backpropagation (spatio-temporal for SNN layers). This approach demonstrates benefits for the CIFAR-10 classification task, for which the hybrid version yields 84.98% accuracy (+2% compared to the full ANN) using 25 timesteps with a VGG-9, while showing x4 higher efficiency than the full ANN in terms of MAC/AC operations (using the energy values in 32-bit 45nm technology from Horowitz, 2014). However, the benefits are smaller on Imagenet (x1.3

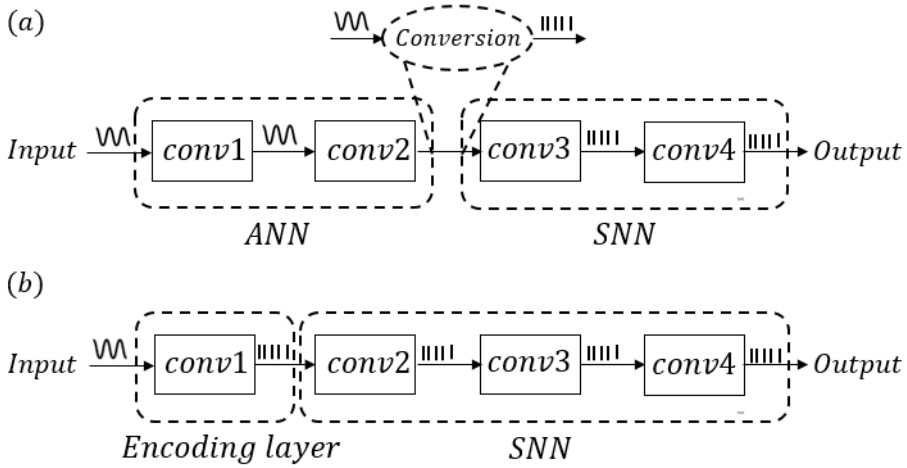


FIGURE 2.7: (a) Example of ANN-SNN hybridization. Here, the first convolutions are done in ANN mode (using high precision activations and MACs operations) and the last convolutions are performed in SNN mode (ACs operations with spikes). A conversion from analog values to spikes is performed between ANN and SNN layers. (b) Encoding layer. The first layer is hybrid ANN-SNN, as synapses perform MAC operations between weights and real-valued inputs, but neurons are spiking. This layer allows the conversion from analog values to spikes.

efficiency), as they could not use more than two spiking layers in the VGG-13 architecture to achieve satisfactory accuracy (-3% compared to the full ANN).

Tandem learning. Another strategy, proposed in Wu et al., 2021, is to couple each SNN layer with an ANN layer with weight sharing. In the training phase, the inference is performed by the SNN and the obtained spike counts are used as activation values by the ANN to perform the backpropagation. Therefore, the offline training phase is accelerated and requires less memory because the backpropagation is done on the ANN (thus removing the need for BPTT). The obtained SNN yields 90.98% accuracy on CIFAR-10 with 8 timesteps (7-layer VGG topology), and 50.22% on ImageNet with 10 timesteps (AlexNet topology). In comparison, the ANN versions achieve 91.77% and 57.55%, respectively. In addition, they report a sparse spiking activity (less than 0.4 spike per neuron per inference on the CIFAR-10 task), and hence up to x20 higher energy efficiency than an equivalent ANN, in terms of MAC/AC operations.

Conversion and direct training hybridization. A hybrid approach between ANN-to-SNN conversion and supervised direct training is proposed in Rathi et al., 2020. Indeed, the ANN-to-SNN conversion yields very good accuracy but at the cost of a high number of inference timesteps, while SNN supervised training can lead to a lower number of timesteps but the spatio-temporal training with BPTT is expensive. Taking the best of both worlds, they used ANN-to-SNN conversion as a pretraining and further fine-tune the SNN with BPTT. For instance with a VGG-16 architecture on CIFAR-10, after the ANN pretraining (250 training epochs), the SNN converged with 20 training epochs, showing the effectiveness of the pretraining. Therefore, the total training duration is reduced to 28 hours (using Nvidia GeForce RTX 2080 Ti TU102 GPU with 11GB memory) compared to 325 hours with SNN training from scratch. However, the memory requirements for training are not reduced, as the SNN still requires training with BPTT. Li et al., 2021 also use this training hybridization technique to calibrate

the parameters but also the initial membrane potentials, in addition to time-varying thresholds. These improvements allow to increase the accuracy with a limited number of timesteps.

Leveraging the Specificity of SNNs

The methods to improve the supervised training proposed previously are mostly inspired by ANNs training. However, the SNN model has some specificities that do not exist in the ANN version.This section shows that, by making efficient use of the rich dynamics of the SNN model, further benefits can be expected in terms of accuracy and efficiency (latency and sparsity).

Leak and threshold of spiking neurons. Typical parameters of spiking neurons such as leak (for LIF neurons) and threshold are usually defined as hyperparameters and not considered in the training phase. However, neuron's leak and threshold are important parameters determining the SNN behavior. For a given set of weights, the threshold determines how much the input neurons must spike in order for the neuron to spike. The leak parameter controls how close to each other input spikes must be for a temporal coincidence to be detected. Thus, including leak and threshold as parameters in the training process can allow a better optimization of the SNN model. For instance, Fang et al., 2021b propose to learn the leak parameters and report higher accuracy than previous spiking approaches on neuromorphic datasets. In Rathi et al., 2021b, both leak and threshold parameters are learned. Note that those added parameters are shared between neurons in a layer, thus the impact on the total number of parameters in the model is negligible. They show an accuracy improvement, compared to learning only the synaptic weights, of about 1% on CIFAR-10 and up to 5% on CIFAR-100 and ImageNet, as well as a lower number of spikes per inference. Moreover, with iso-accuracy tuning the thresholds leads to a reduction in timesteps from 25 to 15 and in spike rate from 1.94 to 1.47 spike per neuron per inference on CIFAR-10. Tuning the leaks further reduces the number of timesteps from 15 to 5 and the spike rate from 1.47 to 0.39.

Synapse dynamics. The choice of the synaptic kernel function, when continuous synapses (rather than instantaneous) are used, is another parameter to explore in order to optimize the SNN model. For instance, Fang et al., 2020a propose a SNN with LIF neurons where the synapses are described as second order infinite impulse response filters, allowing to model various types of kernel (e.g. instantaneous, exponential, alpha, dual-exponential). The coefficients of the synapse filter are jointly learned with the synaptic weights using BPTT. They demonstrate 96.09% accuracy on DVSGesture.

Surrogate gradient. Most backpropagation-based direct training approaches use a surrogate gradient to approximate the derivative of the spiking activity. The derivative of sigmoid functions is often used, as the sigmoid function can be seen as a smooth approximation of the step function (see Fig. 2.5), but other surrogate derivatives, such as exponential or piece-wise linear functions, can also be used (Neftci et al., 2019). While the training performance is robust to the shape of the surrogate function, it is strongly affected by its scale, according to Zenke et al., 2021. For instance, Ledinauskas et al., 2020 show that, by tuning the scale of the surrogate gradient function, the variance of the gradients can be preserved through the layers, avoiding exploding and vanishing gradients.

2.3.2 Accuracy-Latency Trade-off

As SNNs distribute the information through binary events over time, there is an inherent trade-off between accuracy and latency (and hence the energy consumption). Indeed, the latency directly impacts the accuracy, as the precision of the coding depends on the number of timesteps used for inference, as shown in Diehl et al., 2015; Sengupta et al., 2019; Han et al., 2020b. In order for SNNs to replace ANNs for efficient inference on neuromorphic hardware, the accuracy-latency trade-off should be carefully considered. In particular, parameters such as the coding, training strategies, and network architecture width, impact this trade-off. State-of-the-art approaches in conversion, direct training with BPTT, and hybrid training, are compared in terms of accuracy-latency trade-off with regards to these parameters in Table 2.3.

First, the training strategy appears to have an effect on the accuracy-latency trade-off. For instance, the conversion with temporal switch coding proposed in Han et al., 2020a yields better results than the conversion with rate coding of Han et al., 2020b when a reduced number of timesteps is used (256). This is explained by the use of a better threshold balancing. Moreover, fine-tuning with BPTT after the rate-coded conversion in Rathi et al., 2020 improves the accuracy by 1% on CIFAR-10 and by 3% (resp. 5%) on ImageNet when using a VGG (resp. ResNet) architectures with the same number of timesteps (250).

Considering the network architecture, as mentioned in Section 2.3.1, increasing the width can improve the accuracy-latency trade-off. For instance, we observe that the gap between the SNN and ANN accuracy decreases as much as the architecture width increases. This explains the bigger differences between SNN and ANN accuracy on ImageNet with ResNet-34 architecture (21M parameters) than with VGG-16 (138M parameters). The best accuracy among SNNs for a 34-layer ResNet is 67.05% with the spatio-temporal approach (trained with BN) of Zheng et al., 2021 by using a wide ResNet (≈85M parameters). The problem seems partly mitigated by Fang et al., 2021a with a modified implementation of the spiking ResNet achieving 67.04% accuracy with 4 timesteps using the original ResNet-34 architecture. However, both SNNs are still far from the ANN ResNet-34 accuracy (73.31%). Similarly in conversion approaches from Han et al., 2020b; Han et al., 2020a, using the ResNet-34 architecture on ImageNet with a reduced number of timesteps (256 vs 4096), the degradation in accuracy is larger than the one observed for VGG-16. This highlights the accuracy-size trade-off in SNNs: the loss in accuracy due to the quantization of information (which is further increased by reducing the number of inference timesteps) is compensated by increasing the number of neurons in each layer of the network. Notably, tackling the inherent quantization error of SNNs can mitigate the accuracy-size trade-off, as shown in Li et al., 2021; Li et al., 2022, where the accuracy on ImageNet of ResNet and VGG networks is comparable.

In addition, the encoding layer significantly impacts the accuracy-latency trade-off for both hybrid conversion and direct training approaches. Indeed, when compared to the same architecture (ResNet-20 large or VGG-16) with the same training methodology (i.e. conversion with fine-tuning), the encoding layer can allow to reduce the number of timesteps from 250 to 5 with only 1 to 2% accuracy loss, and by optimizing the leak and threshold parameters they further increase the accuracy Rathi et al., 2021b. In addition, direct training approaches yielding high accuracy with very few timesteps (5 to 12), such as Zheng et al., 2021; Fang et al., 2021b; Wu et al., 2019b; Zhang et al., 2020, use the encoding layer. Rathi et al., 2021b study the effect of the input encoding

TABLE 2.3: Impact of input encoding, training and network architecture width on the accuracy-latency trade-off

Paper	Topo.	Encoding layer	Training	Timesteps	Acc. (%)
		CIFAR-10			
Han et al., 2020b	ResNet-20	×	CONV-R	2048	91.36
Han et al., 2020b	ResNet-20	×	CONV-R	256	89.37
Han et al., 2020a	ResNet-20	×	CONV-T	2048	91.42
Han et al., 2020a	ResNet-20	×	CONV-T	256	90.10
Bu et al., 2022b	ResNet-20	✓	CONV-R	16	91.62
Rathi et al., 2020	ResNet-20 (L)	×	CONV-R	250	91.12
Rathi et al., 2020	ResNet-20 (L)	×	CONV-R + BP	250	92.22
Rathi et al., 2021b	ResNet-20 (L)	✓	CONV-R + BP	5	90.29
Lee et al., 2020a	ResNet-11 (L)	×	BP	100	90.95
Zheng et al., 2021	ResNet-19 (L)	✓	BP	6	93.16
Ledinauskas et al., 2020	ResNet-11 (L)	×	BP	20	90.20
He et al., 2016	ResNet-20	/	ANN BP	/	91.25
Rathi et al., 2021b	ResNet-20 (L)	/	ANN BP	/	92.79
		ImageNet			
Han et al., 2020b	ResNet-34	×	CONV-R	4096	69.89
Han et al., 2020b	ResNet-34	×	CONV-R	256	≈20
Han et al., 2020a	ResNet-34	×	CONV-T	4096	69.93
Han et al., 2020a	ResNet-34	×	CONV-T	256	55.65
Bu et al., 2022b	ResNet-34	✓	CONV-R	64	72.35
Rathi et al., 2020	ResNet-34	×	CONV-R	250	56.87
Rathi et al., 2020	ResNet-34	×	CONV-R + BP	250	61.48
Li et al., 2021	ResNet-34	✓	CONV-R + BP	64	71.12
Li et al., 2021	ResNet-34	✓	CONV-R + BP	256	74.61
Zheng et al., 2021	ResNet-34	✓	BP	6	63.72
Zheng et al., 2021	ResNet-34 (L)	✓	BP	6	67.05
Fang et al., 2021a	ResNet-34	✓	BP	4	67.04
Pytorch torchvision models 2021	ResNet-34	/	ANN BP	/	73.31
Han et al., 2020b	VGG-16	×	CONV-R	4096	73.09
Han et al., 2020b	VGG-16	×	CONV-R	256	48.32
Han et al., 2020a	VGG-16	×	CONV-T	2560	73.46
Han et al., 2020a	VGG-16	×	CONV-T	256	69.71
Bu et al., 2022b	VGG-16	✓	CONV-R	64	72.85
Rathi et al., 2020	VGG-16	×	CONV-R	250	62.73
Rathi et al., 2020	VGG-16	×	CONV-R + BP	250	65.19
Li et al., 2021	VGG-16	✓	CONV-R + BP	64	70.69
Li et al., 2021	VGG-16	✓	CONV-R + BP	256	74.23
Rathi et al., 2021b	VGG-16	✓	CONV-R + BP	5	69.00
Pytorch torchvision models 2021	VGG-16	/	ANN BP	/	73.36

CONV-R/T: conversion with rate/temporal coding, BP: backpropagation. Number of parameters: ResNet-20: 0.27M. ResNet-20 (L): 11M. ResNet-19 (L): 13M. ResNet-11 (L): 18M. ResNet-34: 21M. ResNet-34 (L): 85M. VGG-16: 138M.

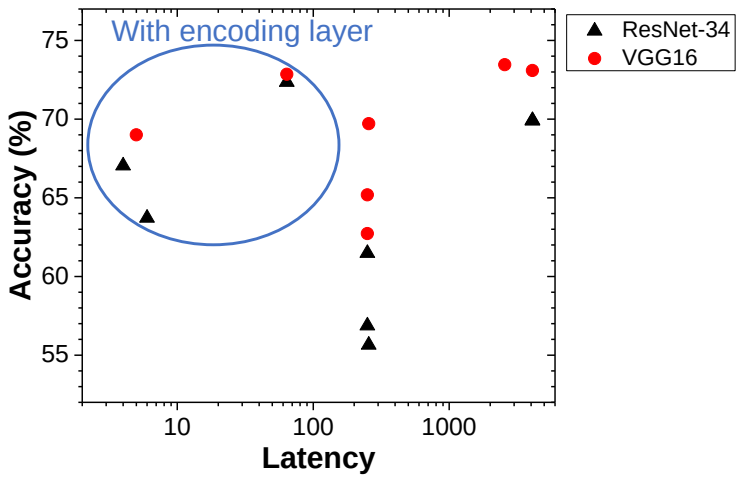


FIGURE 2.8: **Impact of encoding layer and network topology on the accuracy-latency trade-off**, using SNN algorithms from Table 2.3 on the ImageNet dataset.

on both the spike rate and latency for the hybrid conversion with fine-tuning approach (with VGG-16 on CIFAR-10). Using an encoding layer instead of probabilistic sampling induces a significant improvement in the latency (from 150 to 25 timesteps) and in the average spike rate (from 26 to 1.94). The impact of using an encoding layer and a wide network topology on the accuracy-latency trade-off is illustrated in Fig. 2.8.

Hence, it seems that the training strategy, the network architecture width and the use of an encoding layer impact the accuracy-latency trade-off. In particular, the use of wide architectures and encoding layer seems to be the key to compensate for the quantization process inherent to the spike coding with a limited number of timesteps and thus to reach the best accuracy-latency trade-off. However, they bring additional costs. Indeed, the encoding layer requires to use MACs operations (as activations are in high precision, as opposed to spiking), and wide architectures increase memory requirements and area. While the impact of the encoding layer on the SNN energy footprint may be relatively small (as it affects only the input layer), the impact of wide network topology may be more significant.

2.3.3 Estimating Energy Efficiency

Estimating the energy efficiency of SNNs is important in order to ensure that they are more efficient than ANN equivalents, in the context of energy-constrained applications. Many works on SNN algorithms (such as Han et al., 2020b; Rathi et al., 2021b; Fang et al., 2021a; Bu et al., 2022b) focus on decreasing the number of inference timesteps, as a way to decrease latency and hence energy consumption, as seen in Section 2.3.2. However, the latency and energy consumption in event-based implementations are not directly linked to the number of simulation timesteps, but rather to the number of spikes, as will be seen in Chapter 3. Therefore, reducing the number of timesteps is a first step towards SNN efficiency, as it likely decreases the number of spikes per inference (as neurons are limited to fire at most one spike per timestep). However, it does not guaranty a more energy-efficient SNN hardware implementation.

Alternatively, many works (such Panda et al., 2020; Lee et al., 2020a; Rathi et al., 2021b; Wu et al., 2021; Yin et al., 2021) compare the energy efficiency of SNNs relatively to ANNs considering the dynamic energy consumption associated with synaptic operations (ACs for SNNs vs. MACs for ANNs). Indeed, the number of operations is an algorithmic metric that can be easily computed and is agnostic of the hardware implementation. Using this metric, SNNs are often considered more energy-efficient than their ANN equivalents, as ACs consume much less energy than MACs (the exact factor depending on the data precision and the technological node according to Horowitz, 2014). Nevertheless, most of the energy consumption of neural networks in specialized architectures does not come from the arithmetic operations, but from the associated memory accesses (Horowitz, 2014). Therefore, realistic comparisons of ANNs and SNNs should account for the energy of memory accesses and not only of the compute operations. However, the number of memory accesses and their associated energy consumption depend on the underlying hardware implementation. Indeed, the energy footprint of memory accesses does not only depend on the number of operations, but also on the optimization of the dataflow and the ability to exploit data sparsity (Sze et al., 2020). Hence, a realistic comparison of ANNs and SNNs, accounting for memory accesses, requires to make some assumptions on the choices of implementation.

Few studies, such as Khacef et al., 2018; Davidson et al., 2021; Lee et al., 2021a; Lemaire et al., 2022, have compared ANNs and SNNs considering realistic implementations in specialized accelerators. Nevertheless, Khacef et al., 2018; Lee et al., 2021a take the MNIST task as reference to compare ANN and SNN implementations, which is not representative of more difficult tasks (e.g. in terms of sparsity, accuracy, network topologies and possibilities of design optimization). Moreover, Lee et al., 2021a base their comparison on a specific ANN accelerator, and hence the conclusions may not be valid for other accelerators. Davidson et al., 2021 propose an analytical model of the the dynamic energy consumption in ANNs and SNNs (with IF model) considering computations and memory accesses. This models allows to determine the sparsity level required in a SNN (IF) to be more energy-efficient than an equivalent ANN. Lemaire et al., 2022 also propose an analytical model of the the dynamic energy consumption of ANNs and SNNs (with LIF model) considering computations, memory accesses, as well as memory addressing. As Davidson et al., 2021, they conclude that the sparsity level of SNNs is the most important factor determining their efficiency. Notably, they propose a benchmark of ANNs and SNNs on three different datasets (static, dynamic, and event-based), and show that the sparsity achieved by the SNNs make them 6-8x more energy efficient than their ANN equivalent, according to the model. However, Khacef et al., 2018; Davidson et al., 2021; Lemaire et al., 2022 do not consider optimized ANN implementations, which can highly benefit from the optimization of the dataflow and the exploitation of data sparsity, as shown in Sze et al., 2020. Moreover, Khacef et al., 2018; Davidson et al., 2021; Lee et al., 2021a; Lemaire et al., 2022 do not consider the different variants of the IF model frequently used in state-of-the-art SNN algorithms, although the different SNN models require different number of operations and memory accesses, and hence may have a different energy efficiency.

2.4 Conclusion

Between neurosciences, machine learning and neuromorphic hardware, SNNs are defined in many ways, aiming at biological plausibility or high performance. In the context of machine learning, variants of the Integrate-and-Fire model are used with a rate-based or a latency-based spike encoding. SNNs can be used for any application with the objective of high-efficiency inference. In addition, they are particularly suitable for processing event data produced by neuromorphic sensors. Indeed, event-based implementations of SNNs on neuromorphic hardware have been demonstrated to achieve high energy-efficiency. Using analog hardware to efficiently implement some (or all) of the SNN components could lead to further gains.

Furthermore, training strategies specific to SNNs have been reviewed. Among them, biologically-inspired and backpropagation-free learning rules are promising for on-chip training. However, targeting the highest accuracy and efficiency during inference, indirect training (ANN-to-SNN conversion) and direct training with backpropagaton currently lead to the highest performance. In particular, direct training based on backpropagation seems to better leverage the spatio-temporal dynamics of SNNs, allowing to achieve high efficiency (in terms of latency and sparsity). Among direct training methods, single-spike approaches appear promising due to their potentially high sparsity associated with the latency coding. However, in practice, the sparsity reached with this training strategy is not higher than rate-coded approaches and they are hardly compatible with spatio-temporal data. Spatial approaches are the most efficient in terms of computation and memory during training. However they have limitations similar to conversion approaches, as they do not allow to optimize the temporal dynamics of SNNs. Although spatio-temporal training is costly due to the use of BPTT, these approaches have demonstrated so far the highest accuracy with low latency and high sparsity during inference. Moreover, they are naturally compatible with spatio-temporal data, such as data produced by neuromorphic sensors. Therefore, in this thesis, a spatio-temporal approach was chosen to train SNNs in order to achieve high accuracy and efficiency, in particular for processing spatio-temporal data.

In addition, strategies for improving the accuracy and efficiency (in terms of energy consumption and latency) of SNNs have been analyzed. For instance, taking inspiration from ANNs training, as well as optimizing parameters specific to SNNs, can lead to higher accuracy and efficiency. Moreover, the accuracy-latency trade-off in SNNs should be carefully considered. For instance, the use of an encoding layer (i.e. bypassing the spike conversion process for non-spiking data) and wide network architectures can significantly improve both accuracy and latency. In particular, the encoding layer has a relatively small impact on the energy footprint of SNNs, while its impact on accuracy is significant. However, wider network architectures increase both the energy consumption and memory requirements.

Finally, comparing the energy efficiency of ANNs and SNNs considering their implementation on specialized neural network accelerators is not straightforward. Therefore, this comparison must be further investigated, in the objective of using SNNs as alternatives to ANNs for energy-constrained applications.

Chapter 3

Energy Efficiency of Spiking vs. Artificial Neural Networks

3.1 Introduction

ANN inference being highly resource-hungry, designing efficient implementations of ANNs is essential, as seen in Section 2.1.2. In particular, the design of efficient neural networks accelerators is currently an important research topic. From the algorithmic perspective, another approach is to look for more efficient algorithms to replace ANNs, such as SNNs. In digital implementations, ANNs, use multiply-and-accumulate (MAC) operations between input activations (*iacts*) and weights on a layer-by-layer basis. Conversely, SNNs are based on accumulate (AC) operations due to the binary nature of their activations, and present a high sparsity of activations (spikes). However, while SNN event-based implementations naturally benefit from spike sparsity, ANN implementations present other advantages. Indeed, ANNs can also leverage the sparsity of *iacts* via data compression and logic to skip unnecessary MAC operations, according to Chen et al., 2019. Moreover, an efficient dataflow (i.e. scheduling of ANN computations and mapping across Processing Elements (PEs), according to Sze et al., 2020), can optimize the data reuse. Indeed, instead of reading each data required for each computation in an external memory, which is very costly (Horowitz, 2014), data reuse consists in storing locally some data which will be reused in several computations. Optimizing the dataflow to maximize data reuse allows to minimize memory accesses to a distant memory and hence reduces the global energy consumption and latency. Unfortunately, event-based implementations of SNNs cannot leverage data reuse due to the non-flexible and non-predictable order of computations driven by spikes. Hence, the hypothetical advantage of SNNs over ANNs in terms of energy efficiency is not obvious.

Nevertheless, as shown in Section 2.3.3, a realistic comparison of ANN and SNN hardware implementations on dedicated neural network accelerators is currently missing. Indeed, most of the works on SNN algorithms focus on metrics that only partially reflect the energy efficiency, such as the number of timesteps, or the energy consumption of compute operations. Moreover, studies relying on a model of ANN and SNN hardware implementations often lack fidelity or generality considering the ANN implementation.

Therefore, in this chapter, a high-fidelity model is proposed for comparing the energy efficiency of ANN and SNN hardware implementations on neural network accelerators, considering how data reuse and sparsity play a role. Guidelines for improving

the energy efficiency of ANN and SNN implementations are also provided.

Note that the energy efficiency will be measured considering the dynamic energy consumption. Hence throughout this chapter, "energy" will refer to dynamic energy consumption.

This chapter is organized as follows:

- In Section 3.2, the scope of the study, in particular the architecture of neural network accelerators that is considered, is presented.
- In Section 3.3, a model of ANN and SNN dynamic energy consumption is proposed, considering different variants of the IF model for SNNs, which is used to evaluate the energy efficiency of state-of-the-art SNN algorithms.
- In Section 3.4, different ANN models considering data reuse and exploitation of the sparsity are proposed. An upper bound on ANN energy efficiency relative to SNN is provided, which is independent of the hardware implementation, by computing the maximum benefit of data reuse and exploitation of sparsity. A lower bound, representative of a naive ANN implementation, is used as a baseline. Then, these theoretical bound are compared with the state-of-the art ANN accelerators Eyeriss v1 and v2 from Chen et al., 2017; Chen et al., 2019.
- In Section 3.5, the effectiveness of hybrid ANN-SNN implementations is discussed, as an alternative to either fully-ANN or fully-SNN implementations.

The results of this chapter have been published in Dampfhoffer et al., 2023b.

3.2 Scope of the Study

3.2.1 Architecture of Neural Network Accelerators

We consider efficient digital implementations of neural network accelerators based on near-memory computing (as described in Section 2.1.2). Note that the methodology can be adapted to introduce analog elements, considering mixed-signal implementations. For instance, NVMs could replace on-chip SRAMs, modifying the energy associated with memory accesses, and hence the numerical applications should be adapted.

The neural network hardware architecture can either be spatially expanded (each neuron is physically represented) or spatially folded (time-multiplexed), according to Khacef et al., 2018, as depicted in Fig. 3.1. In a spatially expanded architecture, all the memory is on-chip in buffers (such as SRAMs) close to the PEs. In a spatially-folded architecture, which is typically used for ANNs (for example in Eyeriss from Chen et al., 2017), the chip is smaller, with only one buffer on-chip, and there is an off-chip memory (such as DRAM). This type of architecture increases the dynamic energy consumption due to the off-chip memory accesses, but saves area. Indeed, in spatially expanded architectures, the supported network topologies are limited by the size of the chip.

In the case of SNNs, the choice of hardware architectures depends on the processing mode, whether it is an event-based execution, where all layers are computed in parallel, or an ANN-like execution, where layers are computed one at a time. In event-based execution (such as Merolla et al., 2014; Davies et al., 2018; Moradi et al., 2018),

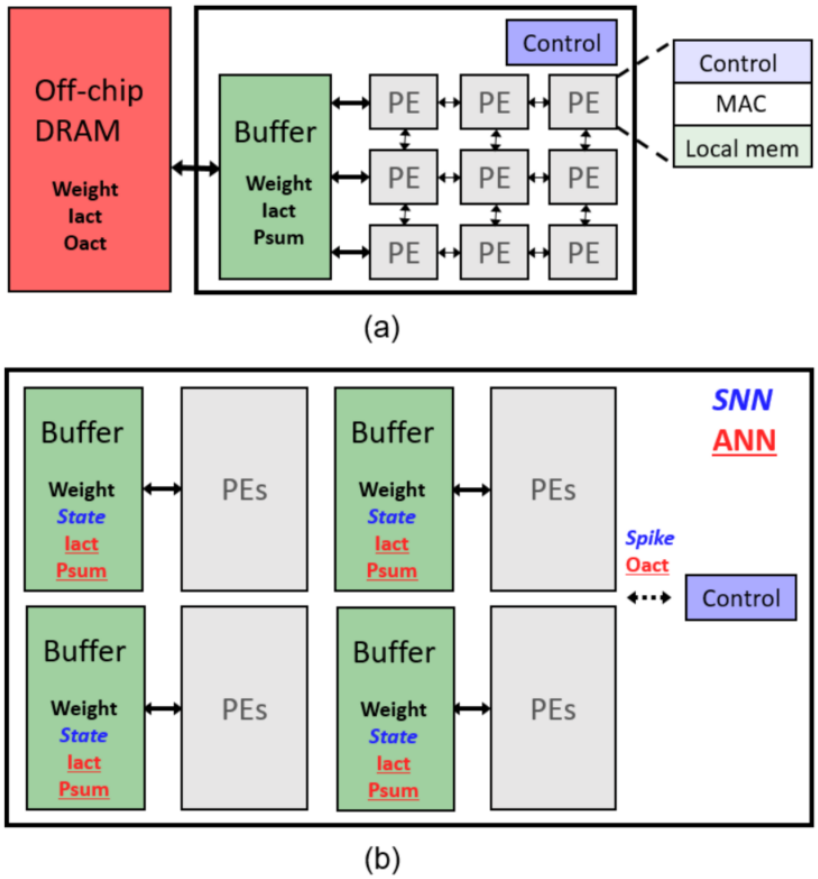


FIGURE 3.1: (a) Spatially folded architecture (such as the Eyeriss v1 chip from Chen et al., 2017). (b) Spatially expanded architecture. Data types specific to ANN and SNN are highlighted and in italic, respectively. Memory buffers on-chip are used to store weights and states (such as membrane potentials or input currents) for SNNs. For ANNs, input activations (iact) and partial sums (psum) (used to store partial results of MAC operations) must be stored in addition to weights.

spikes are encoded in a packet containing the address of the source neuron (whose size depends on the network topology) that is sent to the destination neuron in real time using the AER protocol (Boahen, 2000), as described in Section 2.1.2. Event-based processing allows leveraging the spike sparsity of SNNs, as spikes are not stored and processed immediately, but requires a spatially expanded architecture.

In ANN-like execution, spikes must be stored, but it offers more opportunities for data reuse and allows spatially folded architectures as in ANNs. For instance, Narayanan et al., 2020 propose an output stationary dataflow where spiking neurons are mapped to PEs to minimize the energy consumption of reading and writing the neurons state. This comes at the cost of storing the spikes in a FIFO during the computation of a layer, which must be further sorted to respect the computation order constraints. In Lee et al., 2020b; Lee et al., 2021b, input spikes are stored as ANN activations with only 0 and 1 values, but with an additional temporal dimension (representing SNN temporal dynamics). Therefore, they can process all spikes of a layer in parallel and use a weight stationary or output stationary dataflow. However, they must also store spikes in tensors whose size depends on the temporal resolution. These

approaches provide scalability, but it is unclear whether they provide an energy benefit. Indeed, they lose the advantages of SNNs, i.e. not storing activations and naturally leveraging their sparsity, and still cannot exploit as much data reuse as in an ANN due to the computation order constraints.

Therefore, in this study, we focus on the event-based approach. Instead of taking an existing neuromorphic accelerator, we will consider a more general event-based architecture, and therefore spatially expanded, which is not specific to a SNN model. Moreover, we want to fairly compare ANNs and SNNs only based on the processing mode, independently of the hardware implementation. Therefore, we consider that both use a spatially expanded architecture in order to use the same energy values for memory accesses.

3.2.2 Methods for Evaluating Hardware Efficiency

This study focuses on the applicative case of image recognition applications, as they are frequently used for benchmarking neural networks. On-chip inference (and not learning) is considered. In the models, we take into account only convolutional and fully-connected layers, which represents the main energy consumption of ANNs and SNNs. Activation functions, typically *ReLU* for ANNs and comparison with a constant threshold for SNNs, pooling and normalization layers consume relatively little energy.

We study the dynamic energy consumption associated with the synaptic operations of ANNs and SNNs. The static energy consumption and energy consumption associated with communication are other factors impacting the overall energy consumption of a system. However, dynamic energy consumption associated with synaptic operations being one of the main motivation for the use of SNNs (due to the assumed benefits of spike sparsity and replacement of MACs by ACs), we decided to focus our study on this point. Thus, static energy consumption and energy consumption associated with communication are out of the scope. For the same reason, we focus on energy consumption rather than area, latency and throughput, although they are also important factors to consider when designing an accelerator.

Therefore, the dynamic energy consumption is used as the energy efficiency metric. It is computed using the number of memory accesses (read and write of variables in the memory) and operations (MAC and AC), which are evaluated considering the sparsity and data reuse opportunities.

3.3 Dynamic Energy Consumption of ANNs and SNNs

In this section, the models proposed to compute the dynamic energy consumption of ANNs and SNNs are detailed. These models are based on number of memory accesses and operations (MAC and AC). For ANN, a "naive" model is first presented, which does not consider opportunities for exploiting *iact* sparsity and data reuse, and is used as a baseline. "Ideal" and "realistic" ANN models are presented in the next Section (3.4). Moreover, SNN models are presented considering different SNN variants. Then the naive ANN implementation is compared with the SNN models and numerical applications (based on data from SNN algorithms) are provided.

3.3.1 Naive ANN Model

In ANNs, the output of a neuron is defined as:

$$y_i = \varphi(\sum_j x_j w_{ij} + b_i) \tag{3.1}$$

y_i is the output activation (*oact*) of neuron i, b_i its bias, x_j is the *iact* from presynaptic neuron j, and w_{ij} is the synaptic weight between neurons i and j. φ is an activation function, such as *ReLU*. Therefore, the ANN atomic operation is the synaptic operation, which corresponds to a MAC operation. Note that the number of synapses (N_{syn}) is different from the number of weights (especially in convolutional architectures where weights are reused in multiple synapses). We start by considering a naive ANN implementation : for each MAC, we must read the *iact*, weight and current partial sum (*psum*), and write back the updated *psum* (according to Sze et al., 2020). Therefore, ER_x (respectively EW_x) being the energy of the read (respectively write) operation on the x data, the total energy for the ANN is:

$$E_{ANN} = N_{syn} \times (ER_{iact} + ER_{weight} + ER_{psum} + EW_{psum} + E_{MAC}) \tag{3.2}$$

3.3.2 SNN Models

For SNNs, we evaluate frequently used variants of the Integrate and Fire model (Gerstner et al., 2014). As seen in Section 2.1.1, neurons can be simple Integrate and Fire (IF), or with an additional leak (LIF), meaning that the membrane potential decays over time. Synapses can be instantaneous or continuous, whether the spike is integrated immediately to the membrane potential or is accumulated first in another state variable (input current), which also has a dynamic behavior. Note that, in the case of digital hardware, only the implementation of IF neuron with instantaneous synapse model do not require a temporal discretization (e.g. to apply the leak on the membrane potential at each timestep) and can be fully asynchronous. The different combinations of neurons and synapses are described in the following subsections.

IF neuron and instantaneous synapse (IF+inst)

The dynamics of this model is described by the following equation (setting aside the reset and spiking mechanisms):

$$\frac{dV_i}{dt}(t) = \sum_j \sum_r w_{ij} \delta(t - t_j^r) \tag{3.3}$$

$V_i(t)$ is the membrane potential of neuron i at time t, w_{ij} is the synaptic weight between neuron i and neuron j, t_j^r is the time of the r^{th} spike from neuron j and δ is the Dirac delta function. For each incoming spike from neuron j, we must read the associated weight w_{ij} and the membrane potential (state) of the neuron. In an event-based implementation, we do not need to read the input value, as it is a spike (and hence it communicates directly the addresses of the corresponding weight and neuron state). Then, the weight is simply added to the state with an AC operation. We assume that,

in the case the spikes must be buffered, this energy is included in the communication, which is ignored here. In SNNs, a synapse can receive several spikes. Hence, the energy associated to a SNN synaptic operation must be multiplied by the average number of spikes received per synapse ($N_{spikes/syn}$). We obtain the total energy:

$$E_{IF+inst} = N_{syn} \times N_{spikes/syn} \times (ER_{weight} + ER_{state} + EW_{state} + E_{AC}) \tag{3.4}$$

LIF neuron and instantaneous synapse (LIF+inst)

The dynamics (without spiking and reset) of the *LIF+inst* model is described as:

$$\tau_m \frac{dV_i}{dt}(t) = -V_i(t) + \tau_m \sum_j \sum_r w_{ij} \delta(t - t_j^r) \tag{3.5}$$

This corresponds to an exponentially decaying membrane potential with time constant τ_m. The evolution of the membrane potential with time can be also described in an iterative formulation:

$$V_i^t = V_i^{t-1} \times exp(-\frac{\Delta t}{\tau_m}) + \sum_j w_{ij} \epsilon_j^t \tag{3.6}$$

with ϵ_j^t equals to 1 if neuron j from the previous layer has fired a spike at timestep t, or 0 otherwise. In this model, the states of neurons are updated at each timestep. Therefore, compared to equation (3.4) we must add the energy for updating the state for all neurons (N_{neur}) and all timesteps (T is the number of timesteps in one inference), corresponding to: reading the state, multiplying it with a constant, and writing back the result. We obtain the total energy by combining the operations performed at each incoming spike, from (3.4), and at each timestep:

$$\begin{aligned} E_{LIF+inst} = N_{syn} \times N_{spikes/syn} \times (ER_{weight} + ER_{state} + EW_{state} + E_{AC}) \\ + N_{neur} \times T \times (ER_{state} + EW_{state} + E_{MAC}) \end{aligned} \tag{3.7}$$

Another strategy is to update the states only when necessary, i.e. when there is an incoming spike to the neuron (using an additional variable to record the last spike time), as proposed in Roy et al., 2017. However we found that in recent SNN algorithms, the update at each timestep is more efficient, as $N_{syn} \times N_{spikes/syn}$ is large compared to $N_{neur} \times T$, as will be seen in Section 3.3.4. Moreover, the energy of updating the states when there is an incoming spike is higher than updating at a timestep (as in addition we must compute the total decay from the last to the current input spike).

IF neuron and continuous synapse (IF+cont)

SNNs with temporal coding, for instance Time-to-First-Spike (TTFS) coding, require continuous synapses to track the spike timings (Mostafa et al., 2017). The equations of

the *IF+cont* model (without spiking and reset) are:

$$\begin{aligned} \frac{dV_i}{dt}(t) &= I_i(t) \\ \tau_s \frac{dI_i}{dt}(t) &= -I_i(t) + \sum_j \sum_r w_{ij} \delta(t - t_j^r) \end{aligned} \tag{3.8}$$

$I_i(t)$ is the input current state variable of neuron i at time t. This corresponds to an exponentially decaying synapse current with time constant τ_s. The neuron is IF because the membrane potential only integrates the input current and does not decay over time. We use the discretized formulation in Mostafa et al., 2017:

$$\begin{aligned} V_i^t &= V_i^{t-1} + I_i^t \times \lambda \\ I_i^t &= I_i^{t-1} + \sum_j w_{ij} \epsilon_j^t - I_i^{t-1} \times \lambda \end{aligned} \tag{3.9}$$

Note that the equation of the input current I is similar to that of the membrane potential V for the *LIF+inst* model, using the constant $\lambda \approx \frac{1}{\tau_s}$ to represent the temporal resolution. To update I at each timestep, we must read I, multiply it by a constant, and write back I. To update V at each timestep, we must read V, multiply the previously obtained value of I by a constant and add it to V, and write back V. The update at each spike (instead of each timestep) is not an option for this model due to the continuous synapse. Indeed, an input spike can generate an output spike even after the input spike time, as the membrane potential keeps integrating the continuous postsynaptic potential. Therefore, keeping track of the spike times is highly complex. We obtain the total energy:

$$\begin{aligned} E_{IF+cont} = N_{syn} \times N_{spikes/syn} \times (ER_{weight} + ER_I + EW_I + E_{AC}) \\ + N_{neur} \times T \times (ER_I + EW_I + ER_{state} + EW_{state} + 2 \times E_{MAC}) \end{aligned} \tag{3.10}$$

Moreover, we can compute the energy of the *LIF+cont* model (LIF neurons with continuous synapses combination) from that of the *IF+cont* by adding a MAC operation at each timestep (corresponding to the multiplication of the membrane potential with the decay factor).

3.3.3 Comparison of the Models

We now compare the energy efficiency of the ANN and different SNN models described in the previous section. We choose 8 bit Fixed Point data format as it is commonly used for inference in neural network accelerators (Chen et al., 2019). As explained in Section 3.2, we consider a spatially expanded architecture for both ANNs and SNNs, using SRAMs as on-chip memory, with energy ratios for compute and memory in CMOS 45nm from Horowitz, 2014 (see Table 3.1).

The relative energy consumption of the memory accesses and computations associated with synapse operations (at each spike) and neuron operations (at each timestep) of the ANN and the different SNN models is presented in Fig. 3.2. Note that for the *LIF+inst* and *IF+cont* SNN models, there is an energy associated with the update of

TABLE 3.1: Normalized energy cost relative to a MAC operation.

CMOS Techno. (data precision)	E_{AC}	E_{MAC}	ER/W	ER/W^{reg}
45nm (8 bit) (used in Section 3.3)	0.13x	1x	5.4x	/
65nm (16 bit) (used in Section 3.4)	0.06x	1x	6x	1x

neurons, in addition to synaptic operations, contrary to the *IF+inst*. Thus, the energy breakdown depends on the relative values of $N_{spikes/syn}$, T, N_{syn} and N_{neur}. Here, we use hypothetical values for these variables in order to give a first overview of the energy breakdown of the models. In particular, we assume that $N_{spikes/syn} = 1$ and we use the VGG16 topology to compute N_{syn} and N_{neur}. We consider two cases for T, one with a low latency ($T = 10$) and one with a high latency ($T = 500$).

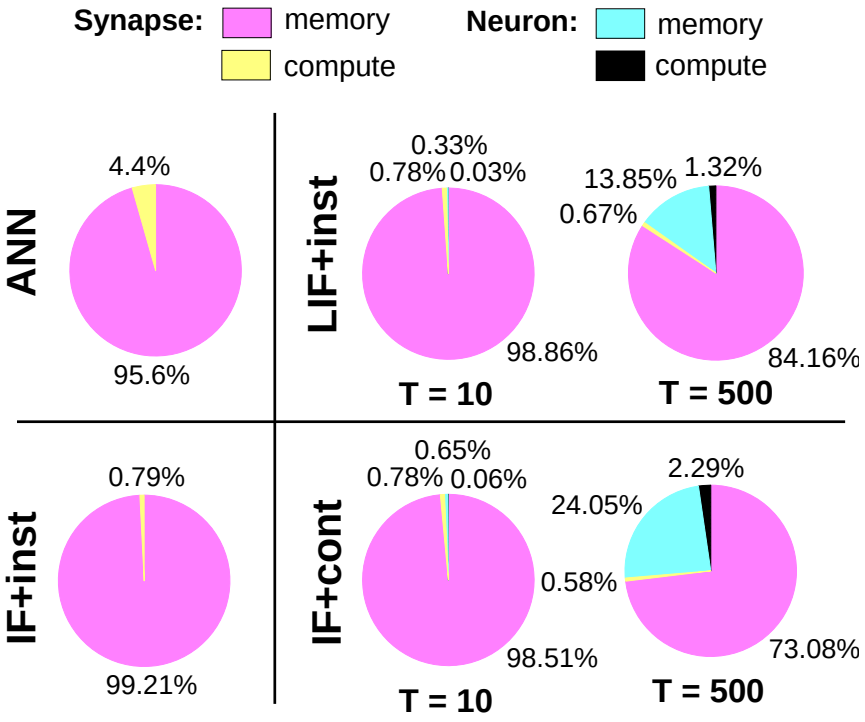


FIGURE 3.2: Relative energy consumption of the memory and compute associated with synapse operations (at each spike) and neuron operations (at each timestep) of the ANN and the different SNN models described in Section 3.3.

We observe that, in all models, the energy cost of compute is very small compared to that of memory. In SNN *IF+inst* compared to ANN, the energy associated with computation is smaller due to the replacement of the MAC by the AC operation. For the *LIF+inst* and *IF+cont* models, the overhead associated with the updates of neurons at each timestep is negligible if the number of timesteps is small (T = 10) but becomes important if the number of timesteps grows (T = 500). In the latter, the energy of updating neurons reaches 15.17% (respectively 26.34%) of the total energy consumption of the *LIF+inst* (respectively *IF+cont*) model.

The relative energy efficiency of SNN models with updates at each timestep compared to ANNs also depends on the ratio N_{syn}/N_{neur}. This ratio depends on the network topology. For instance, in the more compact MobileNet topology, this ratio is

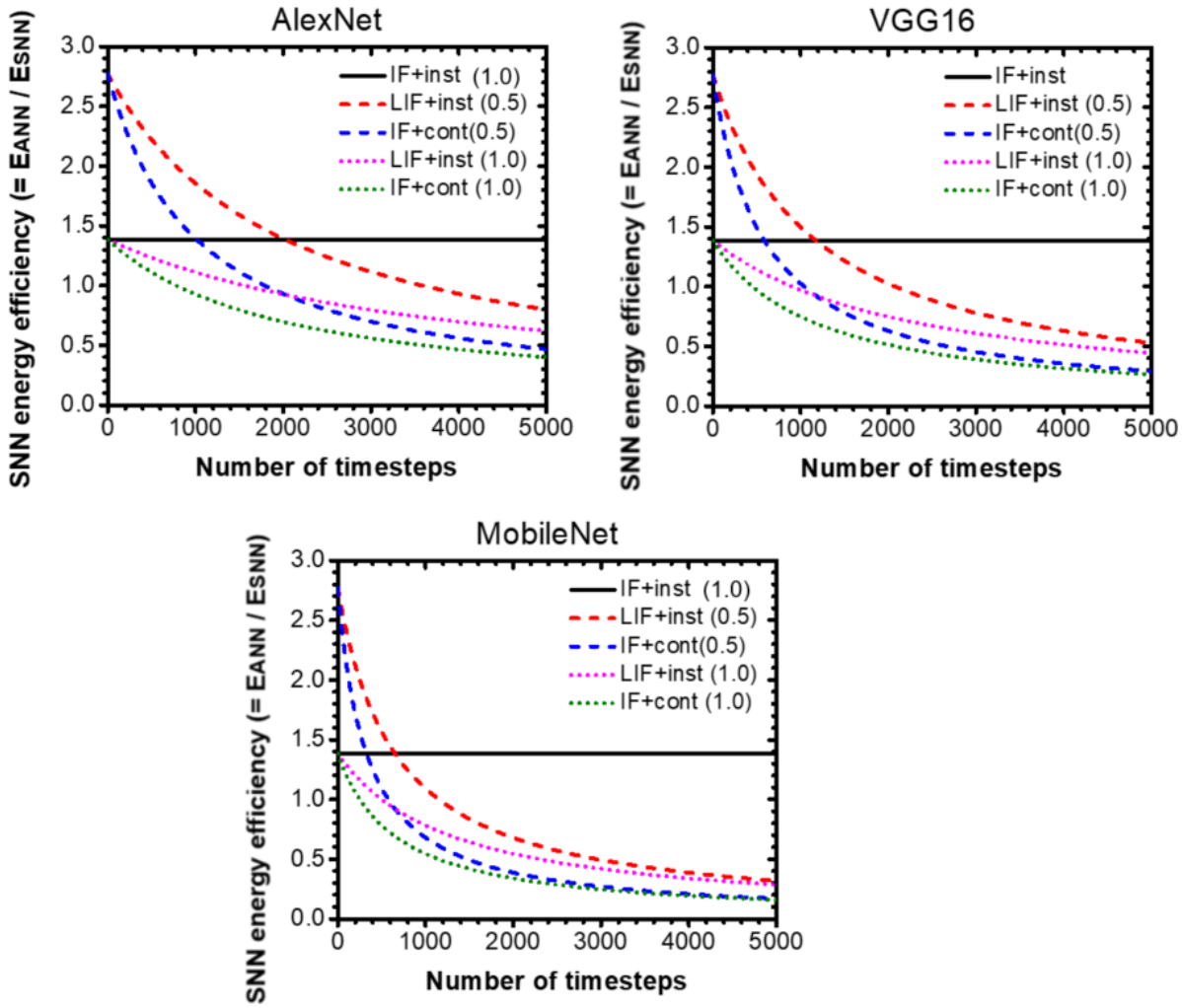


FIGURE 3.3: SNN energy efficiency relative to ANN (=E_{ANN}/E_{SNN}) as a function of the number of timesteps (T), depending on the SNN model and $N_{spikes/syn}$ (in parenthesis). The AlexNet, VGG16 and MobileNet topologies have N_{syn}/N_{neur} = 2.9x10^3, 1.7x10^3, 9.4x10^2, respectively.

3x lower than in the AlexNet topology. The energy efficiency of the different SNN models compared to ANNs (= E_{ANN}/E_{SNN}) depending on the number of timesteps is shown in Fig. 3.3, for a given $N_{spikes/syn}$. The results are shown for the AlexNet, VGG16 and MobileNet topologies, having N_{syn}/N_{neur} = 2.9x10^3, 1.7x10^3, 9.4x10^2, respectively. We see that, in topologies with a low N_{syn}/N_{neur} (such as MobileNet), the energy efficiency of the *LIF+inst* and *IF+cont* models decreases more rapidly with the number of timesteps compared to topolgies with a higher ratio (such as VGG16 or AlexNet). Conversely, the energy of the *IF+inst* does not depend on the update of neurons. Therefore, its relative efficiency compared to ANN is constant with the number of timesteps and does not depend on the size of the topology (but only depends on $N_{spikes/syn}$). In particular, in case of the *IF+inst*, we can compute that $N_{spikes/syn}$ must be lower than 1.38 for E_{SNN} to be lower than E_{ANN} regardless of the network topology. Note that in Fig. 3.3, the SNN sparsity is supposed constant with the number of timesteps (for the purpose of the figure). However, it is likely that, in practice, the number of spikes increases with the number of timesteps. In that case, the relative energy efficiency of

TABLE 3.2: Energy efficiency of state-of-the-art SNNs relative to ANN (= E_{ANN}/E_{SNN}) using models from Section 3.3 and energy ratio in Table 3.1. ANN IS CONSIDERED WITH A NAIVE (NON-OPTIMIZED) IMPLEMENTATION.

Topology	SNN	Acc. (%)	T	$N_{spikes/syn}$	Energy efficiency
			CIFAR-10		
VGG8* [1]	IF+inst	90.98	-	0.30	**4.6x**
VGG16 [2]	IF+inst	90.35	-	1.30	**1.1x**
VGG16 [3]	IF+inst	92.79	-	0.51	**2.7x**
ResNet11 [4]	LIF+inst	90.95	100	3.60	**0.4x**
VGG9 [5]	LIF+inst	90.50	25	0.80	**1.7x**
VGG16* [6]	LIF+inst	92.70	5	0.39	**3.6x**
VGG16 [7]	IF+cont	92.68	680**	0.62	**1.3x**
			ImageNet		
VGG16 [2]	IF+inst	68.93	-	5.00	**0.3x**
VGG16 [3]	IF+inst	72.59	-	1.00	**1.4x**
VGG16* [6]	LIF+inst	69.00	5	0.41	**3.4x**

*with encoding layer. **assumed number of timesteps.
[1] Wu et al., 2021, [2] Han et al., 2020b, [3] Han et al., 2020a, [4] Lee et al., 2020a, [5] Kim et al., 2020, [6] Rathi et al., 2021b, [7] Zhou et al., 2021.

the SNN compared to the ANN will decrease further with the number of timesteps.

3.3.4 Application to SNN Algorithms

We apply the models previously described to investigate the energy efficiency of state-of-the-art SNN algorithms compared to an ANN (see Table 3.1 for the energy ratios used). In this case, the values of $N_{spikes/syn}$, T, N_{syn} and N_{neur} are based on information provided in the corresponding papers. $N_{spikes/syn}$ in a given layer is the average number of spikes fired by a neuron in the previous layer. To obtain an average on the entire network, we need to weight it by the number of synapses in each layer, which is the number of neurons in this layer multiplied by its fan-in (number of input connections). In practice, if the average number of spikes per neuron of each layer is not available, we assume that over the entire network $N_{spikes/syn} \approx N_{spikes/neur}$. Note that some SNN papers, such as Rathi et al., 2021b; Wu et al., 2021, use an encoding layer (the first layer receives real pixel values instead of spikes and therefore does MAC operations as in an ANN) to decrease the number of spikes per inference. In that case, only the energy of spiking layers is considered. Moreover, Zhou et al., 2021 use a temporal coding but no temporal resolution is given. Hence, it is assumed based on Park et al., 2020 using a similar SNN model with temporal coding, achieving comparable accuracy with the same network topology and dataset.

The energy efficiency of SNN relative to ANN (= E_{ANN}/E_{SNN}) for state-of-the art SNN papers is shown in Table 3.2 (using the naive ANN implementation described in this section for comparison and the energy ratios from Table 3.1). We observe that all SNN algorithms have a higher energy efficiency than the corresponding ANN (up to 4.6x more energy-efficient), except in Lee et al., 2020a; Han et al., 2020b (where

$N_{spikes/syn}$ is higher). As in most cases $N_{syn} \times N_{spikes/syn}$ is large compared to $N_{neur} \times T$, the energy of updates at each timestep becomes negligible compared to the energy of synaptic operations. In that case, all SNN models have a similar energy consumption (similar to that of *IF+inst*) and only $N_{spikes/syn}$ determines the energy efficiency compared to the ANN. This will not be the case if the number of timesteps increases, or with a different network topology where the ratio between synapses and neurons is smaller, as shown previously (see Fig. 3.2 and 3.3). Note that even if in these examples, all SNN models have a similar energy consumption, SNN models with an update at each timestep require a time discretization of the inference and the computations are not fully event-based. Moreover, the continuous synapse introduces another state variable for each neuron to store the input current, increasing the memory requirements. In addition, we did not observe a higher accuracy or a higher spike sparsity in the models with leaky neurons or continuous synapses, which could justify their use. For all these reasons, the *IF+inst* model seems a better choice for a digital SNN implementation, in the context of image classification.

In an event-based SNN implementation, no data reuse is possible (due to the non-flexible and non-predictable computations) and spike sparsity is leveraged naturally (due to the event-driven computations). In comparison, we considered a naive ANN implementation (worst case ANN) which does not leverage sparsity and data reuse. Therefore, in the next section, we will consider more favorable ANN models. **In Section 3.4, only the *IF+inst* model is considered**, as it is the more general, and the target SNN sparsity can be computed independently of the network topology.

3.4 ANN Models Considering Data Reuse and Exploitation of Sparsity

In the previous section, we ignored the opportunities to exploit data reuse and sparsity in ANNs, although they improve the energy efficiency. Data reuse (for all kind of data types: weight, *iact* and *psum*) is the number of times a data that has been read once from a distant memory can be reused locally for a MAC operation. The ideal (theoretical limit) data reuse is that each data is only read once from a distant memory and then reused locally in the PEs. In practice, due to hardware constraints, the reuse is never ideal but is optimized with the dataflow. On the other hand, sparsity can be exploited in *iact* and weights, by gating or skipping unnecessary MAC operations (i.e. with a zero operand), and compressing data. In SNNs, *iacts* are already compressed (only non-zero *iacts*, i.e. spikes, are transmitted) and thus the reading of weights and AC are only performed when there is a spike. In ANNs, it requires more logic to process compressed *iact*. Exploitation of sparsity in the weights is not taken into account in this study, as we assume that ANNs and SNNs can process sparse weights and obtain the same benefits. Note that it may be even easier for SNNs to process sparse weights than for ANNs to process both sparse weights and *iacts*. Indeed, the logic would only consist in checking if one operand is zero (skipping of zero spikes is natural), while in an ANN, it has to find the match between two non-zero operands (according to Sze et al., 2020). In this section, we use the energy ratios for memory and compute for CMOS 65nm and the 16 bit Fixed Point data format used in Eyeriss from Chen et al., 2016 (see Table 3.1), in order to compare the energy consumption of SNN with the Eyeriss chip.

3.4.1 Best Case ANN: Ideal Exploitation of Data Reuse and Sparsity

We first investigate the best case for the ANN, corresponding to an optimal data reuse and exploitation of sparsity in the *iacts*. This model gives an upper bound on the ANN energy efficiency relative to the SNN *IF+inst* model, which is independent of the hardware architecture.

Theoretical data reuse, or Reuse Factor (RF), is computed for each layer of a topology given its shape and size. In a fully connected layer, the RF on *iacts* is the number of output neurons, the RF on *psums* is the number of input neurons and there is no reuse of weights (RF=1). In a convolutional layer, the RFs depends on the number of input and output channels, kernel size, image size and stride (details of the formula are given in Putra et al., 2021). The RF of each data types for each layer of AlexNet, VGG16 and MobileNet are shown in Fig 3.4. We observe that theoretical RFs are very high, on the order of 10^2 to 10^3 on average. In recent architectures, such as MobileNet (using depth-wise convolution), there is fewer reuse for every data type.

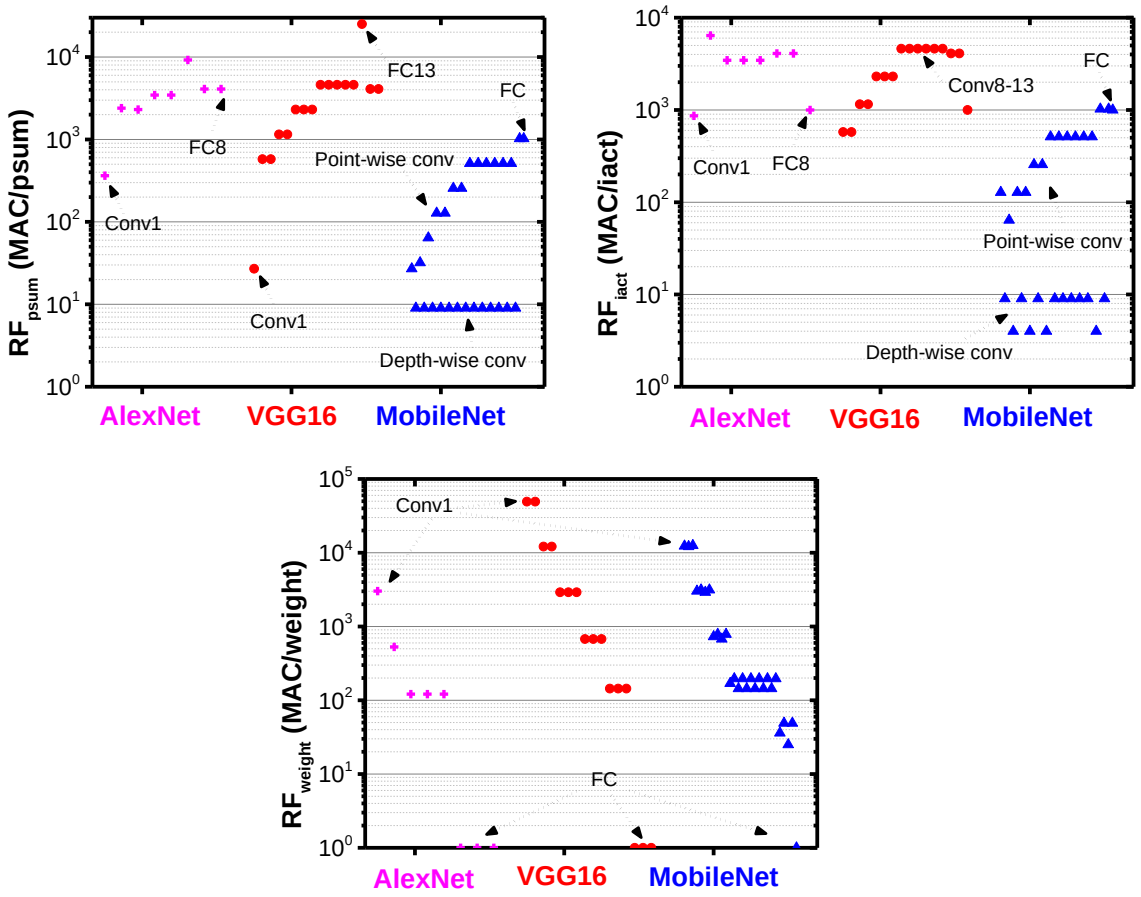


FIGURE 3.4: Ideal data reuse (Reuse Factor) of the three data types (left: *psum*, right: *iact*, bottom: weight) for AlexNet, VGG16 and MobileNet topologies (inspired by Chen et al., 2019). Each point represents a layer of the neural network.

Compared to the previously used ANN energy equation (3.2), we weight the access to a distant memory by the corresponding RF. To reuse data locally, data must be stored in a local storage (such as register files) in the PEs, whose access energy is reduced compared to the distant memory. Then, a data can be accessed from a PE (in the PE

or in a neighbor PE from the PEs array) each time it is reused for a MAC. We add this local storage access (ER^{reg}, EW^{reg}) in the ANN energy:

$$E_{ANN(reuse)} = N_{syn} \times (\frac{ER_{iact}}{RF_{iact}} + \frac{ER_{weight}}{RF_{weight}} + \frac{ER_{psum} + EW_{psum}}{RF_{psum}} + ER^{reg}_{iact} + ER^{reg}_{psum} + ER^{reg}_{weight} + EW^{reg}_{psum} + E_{MAC}) \tag{3.11}$$

This equation gives the energy efficiency of the ANN exploiting the maximum data reuse but not sparsity, if we must store and access each data in a local storage in the PEs (which is the case in most ANN accelerators that are based on systolic architectures, such as Eyeriss from Chen et al., 2017). In that case, the target SNN spike activity (opposite of sparsity) to obtain the same energy efficiency as the corresponding ANN is $N_{spikes/syn} = 0.28$. This is much lower than the target activity obtained with the naive ANN implementation previously described (1.38).

We now consider the ideal exploitation of sparsity in the *iacts*. When there is a zero *iact*, the MAC, the weight read and *psum* read and write in the local memory are saved:

$$E_{ANN(reuse+sparsity)} = N_{syn} \times (\frac{ER_{iact}}{RF_{iact}} + \frac{ER_{weight}}{RF_{weight}} + \frac{ER_{psum} + EW_{psum}}{RF_{psum}} + ER^{reg}_{iact} + (1-\gamma) \times (E_{MAC} + ER^{reg}_{psum} + ER^{reg}_{weight} + EW^{reg}_{psum})) \tag{3.12}$$

γ is the average rate of zero in *iacts*, which is 58% in convolutional layers of AlexNet and VGG16 on the ImageNet dataset (Chen et al., 2017). We did not consider data compression, which can further reduce the energy consumption by reducing the energy of distant memory accesses, as the distant memory accesses are already negligible due to the ideal RFs. Using this equation, the target SNN spike activity becomes $N_{spikes/syn} = 0.15$. This target activity is very low and not achieved in current SNN algorithms as shown in Table 3.2. The target spike activity for the SNN *IF+inst* model to be at least as efficient as the ANN, as a function of the average RF (of all data types) in the ANN (assuming 58% zero *iacts*) is shown in Fig. 3.5. We observe that the target activity decreases rapidly with the ANN RF and becomes small (lower than 0.3) from $RF = 10$.

In practice, such RFs are not achieved due to hardware constraints and exploitation of sparsity requires additional logic consuming energy. Therefore, in the following subsection (3.4.2), we will consider the case of the Eyeriss v1 and v2 accelerators from Chen et al., 2017; Chen et al., 2019.

3.4.2 Real Case Study: the Eyeriss Accelerator

Eyeriss is representative of state-of-the-art deep neural network accelerators with high energy efficiency leveraging both data reuse and sparsity. This allows us to compare the SNN energy efficiency with a realistic ANN hardware implementation.

Eyeriss v1

Eyeriss (Chen et al., 2017) uses a Row Stationary dataflow to increase the data reuse. We use the number of memory accesses given in Chen et al., 2017 to compute the actual

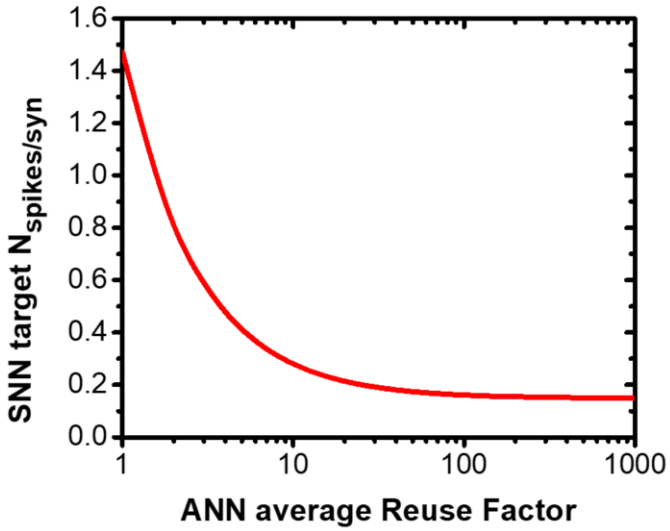


FIGURE 3.5: Target $N_{spikes/syn}$ for the SNN *IF+inst* model to be at least as efficient as an ANN with ideal data reuse and *iact* exploitation of sparsity, as a function of the average data Reuse Factor in the ANN.

RFs. *Iacts*, *oacts* and weights are transferred between the DRAM and PEs through the Global Buffer (GLB), while *psums* are only stored in the GLB. The weights are stored in SRAM in the PEs, which means that they are probably read once in the DRAM and then once in the GLB, to be stored in the PEs. Therefore, we can consider only the number of GLB accesses (as there is no off-chip memory in the spatially expanded architecture). We remove the weights accesses to compute the RFs of *iacts* and *psums*. For this purpose, we compare the number of GLB accesses with the number of accesses required without reuse, as in the naive implementation described in the previous section (each MAC operations requires 4 memory accesses).

We compute the RFs for the AlexNet and VGG16 topologies implemented in Chen et al., 2017 for the ImageNet dataset. For the convolutional layers of VGG16 with batch size of 3, 46.04G MACs are performed in total. This would require 276GB *psums* and *iacts* accesses without reuse (data are encoded in 2B). Instead they perform 11006MB GLB accesses. Therefore the effective average RF for *iacts* and *psums* is 25. Similarly for Alexnet convolutional layers, we obtain an average RF for *iacts* and *psums* of 80. They consider only convolutional layers, which consumes more energy compared to fully connected layers in deep neural networks. They use a batch size superior to 1 to increase the weight reuse, and we consider a batch size of 1, but our computation remains the same (as this does not impact the *psums* and *iacts* reuse). In addition to *iacts* and *psums* accesses in a distant memory, we must consider local data accesses, due to data reuse, from local register files which can be either in the PE (x1 energy cost compared to a MAC) or in neighbor PEs in the PE array (x2) (according to Sze et al., 2017). To simplify, we assume they are accessed in the PE.

Eyeriss leverages *iact* sparsity with data gating logic in the PEs (MAC and weight read are gated when *iact* is zero), which can save 45% of the PEs power consumption. We interpret this as the following: when there is a zero *iact*, the power consumption of PEs is only 55% of the power consumption when the *iact* is non-zero, where the power consumption of the PE corresponds to the associated MAC operation and memory

accesses. The operation is gated but hardware cycles are still spent semi idle. Thus, the obtained energy for a gated operation is only affected by the power as time is constant. This saving includes static energy consumption, which was not considered in our model. As we do not know the relative consumption of static and dynamic factors in Eyeriss PEs, we assume the 45% energy savings corresponds to the savings in the dynamic energy considered here. We obtain the energy of the ANN Eyeriss model:

$$E_{ANN(Eyeriss)} = N_{syn} \times ((1-\gamma) + 0.55 \times \gamma) \times (ER_{weight} + \frac{ER_{iact} + ER_{psum} + EW_{psum}}{RF_{avg}} + E_{MAC} + ER^{reg}_{iact} + ER^{reg}_{psum} + EW^{reg}_{psum}) \quad (3.13)$$

$\gamma = 0.58$ is the average rate of zero *iact* in AlexNet and VGG16. RF_{avg} is the average RF computed for *iacts* and *psums*, which is 25 (resp. 80) for VGG16 (resp. Alexnet). Note that the cost to read weights once from the buffer before storing them in the PEs does not appear in the equation. Indeed, we assume that the associated energy is negligible as it corresponds to the maximum reuse of weights. However, weights are stored in SRAM in the PEs, and hence the energy cost to read them is the same as the cost to read a data in the buffer. Comparing the total energy of the ANN Eyeriss and SNN *IF+inst* models, we get that $N_{spikes/syn}$ in the SNN must be lower than 0.44 (resp. 0.42) for VGG16 (resp. AlexNet) topology, for the SNN to be more energy-efficient than the ANN. This target spike activity is much lower than the one corresponding to the naive ANN implementation (1.38), but higher than the one corresponding to the ideal best case ANN (0.15).

Eyeriss v2

Eyeriss v2 (Chen et al., 2019) is more energy-efficient than the v1 due to a flexible hierarchichal mesh network-on-chip (NoC) and better exploitation of sparsity in PEs. Both *iacts* and weights are compressed and processed by the PEs directly in the compressed form. Therefore, MACs with zero weight or *iact* are skipped (and not gated as in v1). Pruned networks are used to increase the sparsity in weights, improving the energy benefits. Compared to v1, the Eyeriss v2 achieves x3.0 (resp. x1.9) higher energy efficiency with AlexNet (resp. MobileNet v1). The benefits are higher when using a pruned version of the networks (x11.3 and x2.5, respectively).

However, the paper lacks some important metrics for us to compute the corresponding energy equation, for instance GLB accesses are not specified. Therefore, we use the comparison between the two versions, given in the Eyeriss v2 paper, to translate it into the comparison between our models of Eyeriss v1 and v2. We assume that SNNs can exploit the sparsity in weights of pruned network topologies with the same energy benefits, as explained in Section 3.4. Thus, it is fair to compare the previous results on SNNs with the Eyeriss v2 without using pruned networks. Moreover, we did not consider the NoC efficiency in this study. Therefore, if we consider only the benefits due to sparse PEs with non-pruned networks, the energy efficiency is only improved by x1.15 (x1.06) for the AlexNet (MobileNet) topology from the v1 to the v2. Indeed, the *iacts* sparsity alone compensate slightly the overhead of the sparse PEs logic and the compression of non-sparse data. Therefore, we consider that the target

sparsity in the SNN must be x1.15 (for the AlexNet topology) lower compared to the one obtained considering Eyeriss v1, and thus becomes $N_{spikes/syn} = 0.37$.

3.4.3 Summary and Discussion of the Results

Summary of the results. The relative energy consumption of MAC, local memory (accesses in register files) and distant memory (accesses in the buffer) in the different ANN models (baseline, ideal reuse + *iact* sparsity and Eyeriss v1) is depicted in Fig. 3.6. We observe that, in the baseline model (worst case), all the memory consumption comes from a distant memory (96% of the total energy consumption), while in the ideal reuse + *iact* sparsity model (best case), it comes from a local memory (84.06% of the total energy consumption, against 0.32% from memory accesses in a distant memory). Eyeriss v1 is closer to the best case than the worst case, with only 2.2% of the energy consumption due to a distant memory access and 88.02% to a local memory, showing the effectiveness of the dataflow to optimize data reuse.

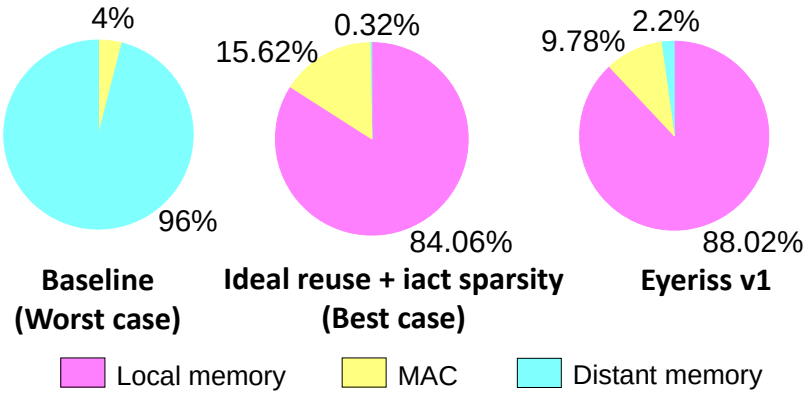


FIGURE 3.6: Relative energy consumption of the local memory, MAC and distant memory in the three ANN models (described in equations (3.2), (3.12) and (3.13), from left to right). AlexNet topology is used in Eyeriss v1 case.

The main results of this study for evaluating the energy efficiency of SNN *IF+inst* model compared to the previously described ANN models are summarized in Fig. 3.7 and Table 3.3. Fig. 3.7 shows the SNN *IF+inst* energy efficiency relative to the ANN ($=E_{ANN}/E_{SNN}$) as a function of $N_{spikes/syn}$ for each ANN model (using AlexNet topology for Eyeriss v1 and v2). Table 3.3 gives the target spike activity for the SNN *IF+inst* model to have the same energy efficiency as the ANN. This corresponds in Fig. 3.7 to the value $N_{spikes/syn}$ in the x-axis of the intersection between the y-axis at SNN energy efficiency = 1 and the curves. We see that above 0.5 spikes per synapse per inference, SNNs cannot compete with ANNs in the realistic (Eyeriss) and ideal cases. However, the SNN energy efficiency grows rapidly as $N_{spikes/syn}$ decreases. For instance, with a spike activity of 0.1, the SNN is 3.6x (resp. 1.5x) more energy-efficient than the ANN implementation of the Eyeriss v2 model (resp. the ideal model), and 7.3x (resp. 3.0x) if the spike activity is 0.05.

Discussion on the choice of energy values. In this study, a different data precision was used as reference for Section 3.3 (8 bit) and Section 3.4 (16 bit). In this case, changing the data precision changes significantly the ratio of the energy consumption

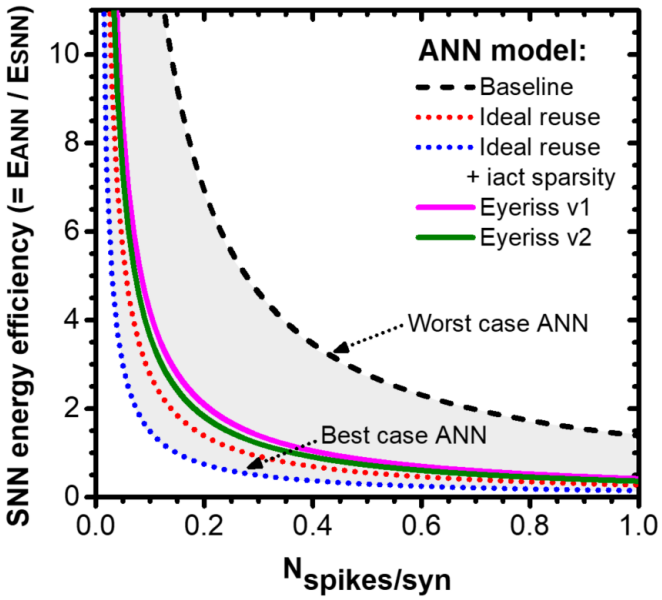


FIGURE 3.7: SNN *IF+inst* energy efficiency relative to ANN (=E_{ANN}/E_{SNN}) as a function of $N_{spikes/syn}$ for the different ANN models considered, using AlexNet topology in Eyeriss v1 and v2 models.

TABLE 3.3: Target spike activity for the SNN *IF+inst* model to be at least as efficient as the ANN depending on the ANN accelerator model

ANN accelerator model (Section)	$N_{spikes/syn}$
MAC vs. AC comparison (state-of-the-art)	*≈ 8-31*
Worst case ANN: Baseline (3.3)	**1.38**
Ideal reuse (3.4.1)	0.28
Best case ANN: Ideal reuse + *iact* sparsity (3.4.1)	**0.15**
Eyeriss v1: reuse + *iact* sparsity (3.4.2)	0.42
Eyeriss v2: reuse + *iact* sparsity + data compression (3.4.2)	0.37

AlexNet topology (non-pruned) is used in Eyeriss v1 and v2 cases.

between AC and MAC operations, but only slightly changes the ratio between the energy consumption of a MAC compared to a memory access (see Table 3.1). As memory accesses dominate the energy consumption in both ANN and SNN, the impact of this choice is limited, and does not change the conclusions of this work. Indeed, with 8 bit data precision, the results from our experiments (see Table 3.3) only change slightly: the target SNN spike activity becomes 1.38, 0.30, 0.16, 0.43, 0.38 (instead of 1.38, 0.28, 0.15, 0.42, 0.37) for the baseline, ideal reuse, ideal reuse + *iact* sparsity, Eyeriss v1, Eyeriss v2, respectively. Conversely, changing the cost of a SRAM access relatively to the cost of a register file access would significantly change the results of the target $N_{spikes/synapses}$ when comparing with an ANN implementation benefiting from data reuse (and hence using register files). In that case, the higher (resp. the lower) the cost of a SRAM access compared to a register file, the lower (resp. the higher) the energy efficiency of the SNN compared to the ANN, as SNN do not benefit from data reuse. The energy consumption of a memory access is related to the memory capacity

(the larger the memory capacity, the higher the energy cost of the memory access). The memory values considered in this Section correspond to the one in the Eyeriss architecture (Chen et al., 2016). This choice could be re-evaluated depending on the hardware implementation and the neural network topology.

Comparison with related works. For comparison, the very naive case of only comparing the MAC and AC operations (associated with synaptic operations) weighted by their energy value, as it is done in many works on SNN algorithms (such as Panda et al., 2020; Lee et al., 2020a; Rathi et al., 2021b; Wu et al., 2021; Yin et al., 2021), is added in Table 3.3. For instance, an AC operation consumes $\approx$ 8x (resp 31x) less energy than a MAC operation in the case of 8-bit (resp. 32-bit) data in 45 nm technology (Horowitz, 2014). Therefore, with $N_{spikes/syn} \approx 8$ (resp. 31), the SNN is considered as efficient as the equivalent ANN. We can see that this estimation is far from reality, when comparing with our hardware-aware models. The two closest studies to ours (Davidson et al., 2021; Lemaire et al., 2022), which also propose an analytical model of the dynamic energy consumption of ANNs and SNNs, come to the same conclusions, that is: SNN spike sparsity is the most important factor determining their energy efficiency compared to ANNs. However, the results obtained are closer to our naive ANN baseline, as they do not consider ANN optimizations such as exploitation of data reuse and data sparsity. Davidson et al., 2021 use an IF model and hence can derive a conclusion independently of the network topology and number of SNN timesteps. They conclude that the SNN should have at most 1.72 spikes/neuron to be as energy-efficient as the equivalent ANN. The difference with our baseline model is mainly that they use different values for computation and memory accesses (in particular, they consider that a multiply operation has the same energy consumption as a SRAM access, which could be discussed). Lemaire et al., 2022 additionally considers the energy consumption associated with memory addressing (i.e. computing the memory address of an element), which was not considered here. Indeed, in ANNs (in a naive implementation with no data compression), the addresses can be computed by only incrementing the index, while for SNN as operations are performed asynchronously, the index must be calculated from scratch at each spike. However it only consists of compute operations, and hence it is shown to have a very small impact on the global energy consumption. In addition, they consider that spikes are stored in a FIFO (which was neglected here), and hence they must be read and written. However, the number of these memory accesses corresponds to the number of spikes produced, which is proportional to the number of neurons, and therefore relatively small compared to the memory accesses related to membrane potentials and weights (proportional to the number of synapses). In addition, they consider different SRAM access energy consumption depending on the memory size, while we have considered a unique SRAM access energy. They use the LIF model and hence derive conclusion based on the network topologies and number of timesteps considered in the study. They conclude that their SNN having 0.08 spikes/synapses is 8x more energy-efficient than its ANN counterpart. According to our results (neglecting the overheads associated with the leaky neuron behavior), this sparsity would rather result in a 2-5x energy benefit (considering ideal and Eyeriss v1 cases). Notably, our naive baseline implementation is more optimistic than theirs, maybe due to the different hyperparameters used (e.g. number of timesteps, network topology, energy values) and the overheads induced by the LIF model.

3.5 Hybrid ANN-SNN Implementations

The results show that SNNs energy efficiency compared to ANNs mainly depends on the SNN spike sparsity. However, we have considered the average SNN spike sparsity at the network level, although it can be very different from one layer to another. Therefore, hybrid ANN-SNN implementations, i.e. a network with ANN and SNN layers, become of interest to leverage the best of both implementations. Indeed, based on the layer-wise SNN spike activity, one could determined for each layer if it would be more optimal to implement it in ANN or in SNN. Note that in between ANN and SNN layers, a conversion from analog values to spike (and vice versa) is required. Hence, the cost of this conversion must also be considered.

The SNN layer-wise spike activity depends on various factors such as the training methodology, network topology and dataset, as shown in Fig. 3.8. In some cases, a pattern observed is that spike activity decreases with the depth of the layer. Therefore, using ANN layers at the beginning of the network, where the spike activity is typically higher, and SNN layers at the end, could lead to more energy efficient implementations in this case. Having only two groups of consecutive layers (ANN or SNN) also avoids multiple conversions. In this case, one must find the optimal separation between the ANN and the SNN using the SNN spike activity at each layer.

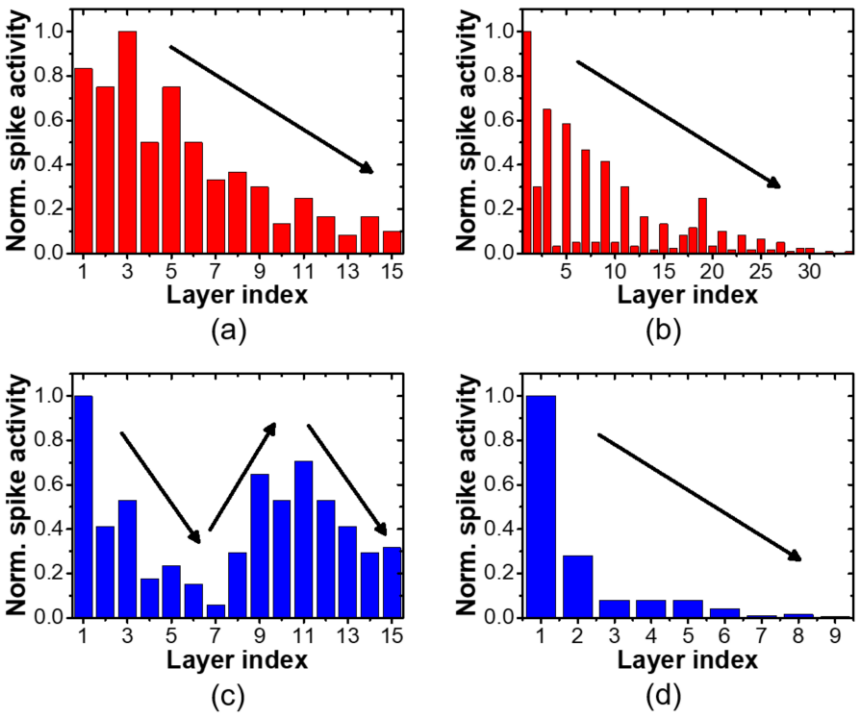


FIGURE 3.8: Normalized layer-wise spike activity of different SNNs with different training methods and datasets. (a) VGG16 from conversion pre-training with spiking backpropagation fine-tuning on ImageNet (Rathi et al., 2021b). (b) ResNet-34 from conversion on ImageNet (Sengupta et al., 2019). (c) VGG16 from conversion pre-training with spiking backpropagation fine-tuning on CIFAR10 (Rathi et al., 2021b). (d) VGG9 with spiking backpropagation training on CIFAR10 (Lee et al., 2020a).

Taking as example the SNN ResNet-34 from Sengupta et al., 2019, using its layer-wise spike activity shown in Fig. 3.8 (but non-normalized), we can compute the potential efficiency of such hybrid ANN-SNN architecture. Note that the spike activity numbers are approximate and therefore the following results are only indicative. With $N_{spikes/syn}$ = 2.4 over the entire ResNet-34 network, the ANN implementation is more energy-efficient than the SNN implementation (6.3x for the Eyeriss v2 model). However, in the last 10 layers of the SNN, the sparsity is much higher ($N_{spikes/syn}$ = 0.18). These layers implemented in SNN are 2.1x more energy-efficient than the corresponding layers in an ANN Eyeriss v2 implementation. Therefore, implementing the first 22 layers in an ANN and the last 12 in a SNN would result in a hybrid ANN-SNN implementation 1.2x more energy-efficient than the ANN Eyeriss v2 implementation. Note that we used the ANN reuse factors and *iact* sparsity in Eyeriss for VGG16, as the ResNet-34 topology was not implemented. The efficiency of hybrid ANN-SNN architectures increases with SNN algorithms having a lower spike activity. For instance, the SNN implementation of the VGG16 proposed in Rathi et al., 2021b (shown in Fig. 3.8a), using an encoding layer, is 1.1x more energy-efficient than the ANN Eyeriss v2 implementation. However, when implementing the first 6 layers in an ANN, the hybrid ANN-SNN implementation would be 1.3x more energy-efficient than the ANN Eyeriss v2 implementation. Indeed, the last 10 layers in SNN implementation are 2.2x more energy-efficient compared to their ANN implementation due to their high spike sparsity ($N_{spikes/syn}$ = 0.21).

However, sparsity and data reuse in ANNs also vary between layers. Therefore, we must take into account these layer-wise factors to evaluate the efficiency of a hybrid ANN-SNN architecture. In addition, the energy associated with the conversion process must also be considered. Hence, the potential of hybrid ANN-SNN implementations should be further investigated.

3.6 Conclusion

Brain-inspired SNN implementations hold the promise of significant energy savings. However, our analysis shows this is contingent on a high level of spike sparsity. This study demonstrates that, contrary to previous thinking, the main advantage of SNNs accelerators compared to ANNs comes primarily from exploiting the sparsity of spikes and not from the replacement of MAC by AC operations. Indeed, memory accesses largely dominate computing operations in terms of energy consumption. Moreover, although SNN models with time discretization (such as LIF neurons or continuous synapses) add some neuronal operations (compared to the ANN and IF neuron and instantaneous synapse model), their impact is relatively small compared to the synaptic operations. Hence, reducing the memory accesses and operations associated with synaptic operations remains essential for an energy efficient SNN accelerator. In addition, the IF neuron and instantaneous synapse model seems to be a good choice among SNN models for image recognition tasks. Indeed, it does not depend on a time discretization, hence allowing a fully asynchronous event-based processing, while showing similar performance in terms of accuracy and spike sparsity for these tasks.

For the first time, a lower and upper bound for the relative energy efficiency of SNN models compared to ANN models is provided. These bounds are based on a naive worst case ANN implementation and a theoretical best case assuming perfect

data reuse and exploitation of sparsity. The SNN energy efficiency compared to ANN implementations in Eyeriss v1 and v2 accelerators was also investigated. However, the results showed that current SNN algorithms do not reach a sufficient spike sparsity at the network level to compete with efficient ANN accelerators such as Eyeriss. Indeed, leveraging spike sparsity in an ultra-light fashion comes at the cost of loosing opportunities of data reuse. Therefore, the spike sparsity must be high enough so that the event-based SNN implementation becomes more efficient than the ANN implementation. Hence, increasing the spike sparsity of SNNs should be further investigated. Moreover, SNN implementations of neural networks offering fewer opportunities of data reuse, such as compact CNNs (e.g. MobileNet) or fully connected topologies, may be particularly competitive compared to ANN implementations. Alternatively, hybrid ANN-SNN architectures appear to be a promising solution to leverage the best of both worlds.

Although we have considered dynamic energy consumption as a metric of efficiency, we believe that the results can be extended to static energy consumption and latency of SNN implementations. Indeed, latency in event-based implementations is also directly related to the sparsity (less spikes to process in event-based manner means faster processing). Moreover, static energy consumption is directly related to latency.

In addition, although described in the case of a fully-digital implementation, the model can be adapted to mixed-signal implementations with a near-memory computing architecture, for instance, using NVMs to store memory on-chip instead of SRAMs. In this case, the conclusion on the energy efficiency of SNNs should be unchanged. Indeed, as memory accesses are dominant compared to compute, the synaptic operations will remain the main source of energy consumption, and hence spike sparsity will still determine the SNN efficiency. However, the exact relative efficiency of ANN and SNN could be modified, depending on the specific choices of design and technology, and hence the numerical applications should be adapted.

Chapter 4

Improving Accuracy and Efficiency of Spiking Neural Networks

4.1 Introduction

As shown in the study of the energy consumption of SNNs (Chapter 3), a high spike sparsity is required to obtain energy-efficient SNN implementations. However, most current SNN algorithms do not reach a sufficient spike sparsity to compete with efficient ANN implementations. We believe that this may be due to the fact that SNNs are mostly benchmarked on static vision tasks, such as image classification, which may not suit them. Indeed, static data must be decomposed using artificial timesteps in order to match the SNN temporal dynamics. Moreover, there is a trade-off between the SNN accuracy and the number of timesteps used (Han et al., 2020b). Therefore, SNNs may rely on a high number of spikes in order to reach competitive accuracy.

Conversely, SNNs have been less considered for spatio-temporal applications (such as audio data), although their temporal dynamics may better fit spatio-temporal rather than static data. Indeed, the data are already sequential, which means that it is not necessary to create artificial timesteps to match the SNN dynamics. Indeed, a SNN can be seen as a form of RNN with only a self-recurrence (from one neuron to itself), instead of a full recurrence (from all neurons to all neurons in the same layer). Moreover, a full recurrence can be added to a SNN, making it a Spiking RNN (SRNN), in order to improve the accuracy on sequential data. In addition, ANN implementations of recurrent topologies enable less opportunities of data reuse than convolutional topologies, as weights are not shared and cannot be reused. Therefore, SNN implementations could provide further benefits. Besides, spiking recurrent topologies demonstrated higher energy and time savings than convolutional topologies on the Loihi neuromorphic chip according to Davies et al., 2021.

This chapter is organized as follows:

- In Section 4.2, a novel SNN model applicable to recurrent topologies inspired by Gated Recurrent Units (GRU), namely SpikGRU, is presented. It is compared to other SNN and ANN models on audio spiking datasets.
- In Section 4.3, we study how the sparsity can be leveraged in recurrent SNNs to further improve their efficiency. For this purpose, the model SpikGRU is compared to GRU on a keyword spotting task in terms of trade-off between accuracy and number of operations per inference.

- In Section 4.4, a more realistic evaluation of the energy efficiency of SNN and ANN hardware implementations with gated recurrent topologies is presented, by adapting the energy model from Chapter 3.

Some of the results have been published in Dampfhoffer et al., 2022; Dampfhoffer et al., 2023a.

4.2 A Novel Recurrent SNN Model: SpikGRU

We believe that SRNNs can show a higher sparsity on spatio-temporal data, such as audio, than SNNs on static data. In addition, we used spiking datasets, in order to improve the energy efficiency of the SRNNs. Indeed, spiking data can directly be fed in SNNs without pre-processing, allowing to leverage their sparsity. Furthermore, gated recurrent networks, such as the Long Short-Term Memory (LSTM, from Hochreiter et al., 1997) and the Gated Recurrent Unit (GRU, from Cho et al., 2014) models, have been proposed to improve the performance of simple RNNs. This motivated us to propose a novel model of SRNN inspired by the GRU: the Spiking Gated Recurrent Unit (SpikGRU). The objective is to propose a model leveraging the accuracy of the GRU and the spike sparsity of SNNs. The novelty of our model, compared to previous propositions of gated recurrent SNNs, is to keep the operations at the neuron level (i.e. cell level in gated recurrent units) in high precision, while leveraging the spike sparsity for the output activations, in accordance with the conclusions of Chapter 3. Indeed, the energy footprint of neuronal operations is proportional to the number of neurons, while the footprint of synaptic operations is proportional to the number of synapses, hence to the square of the number of neurons in recurrent topologies. Therefore, neuronal operations have a low impact on the dynamic energy consumption, and hence, keeping them in high precision does not severely impact efficiency, but bring benefits in terms of accuracy.

In this study, we investigate the performance of LIF, Current-based LIF (Cuba-LIF), our proposed SpikGRU, and ANN models (RNN and GRU) with recurrent topologies. We compare them in terms of accuracy and number of operations on three spiking audio datasets. The datasets are from a DAS (DASDIGITS, from Anumula et al., 2018a) or from a neurophysiology-inspired pre-processing (SHD and SSC, from Cramer et al., 2020), for digits and single words classification.

4.2.1 Models of Recurrent SNNs

Leaky Integrate-and-Fire and Current-based Models

In this chapter, the SNN models are seen as RNNs and simulated with timesteps, and hence they are described using iterative formulations.

The LIF model is commonly used in SNNs for deep learning applications. The LIF model with a recurrent network topology can be described as:

$$v_t^l = \beta \odot v_{t-1}^l + W_v s_t^{l-1} + U_v s_{t-1}^l + b_v - v_{th} s_{t-1}^l \tag{4.1}$$

$$s_t^l = H[v_t^l - v_{th}] \tag{4.2}$$

v_t^l and s_t^l are vectors corresponding respectively to the membrane potential and output spikes of neurons from layer l at time t. $\odot$ denotes element-wise multiplication. Spike firing happens when the membrane potential is superior to the threshold v_{th}, which corresponds to the Heaviside step function H. After each spike, v_{th} is substracted from the membrane potential of spiking neurons. The parameters of the models are W_v and U_v, the weight matrices of feed-forward and recurrent connections (resp.), and b_v, the bias vector. The time constant β corresponds to an exponential decay of v over time.

SNN models with a more sophisticated temporal dynamics than the LIF can achieve superior accuracy for processing temporal data. For instance, Yin et al., 2020; Bellec et al., 2018b; Yin et al., 2021 show that the Adapt-LIF is more accurate than the LIF for speech recognition. In addition to the leaky neuron, the Adapt-LIF uses an adaptive threshold with temporal dynamics (the threshold is increased after each spike fired and decays exponentially with time). In addition, heterogeneous time constant parameters learned per neuron (as opposed to fixed for a layer) can improve the learning on temporal data, allowing the neurons to specialize at different time scales, according to Perez-Nieves et al., 2021. In this work, we focus on the Cuba-LIF model, which is a LIF neuron with continuous exponential synapse. In its iterative formulation, the Cuba-LIF introduces an input current i, which integrates the incoming spikes before transmitting them to v with a time constant α and parameters W_i, U_i and b_i. v_t^l takes as input a linear combination of its previous state v_{t-1}^l and its input i_t^l. Note that, in this work, α and β time constants of LIF and Cuba-LIF models are defined as vectors (different constants per neuron) of trainable parameters, as in Perez-Nieves et al., 2021. We use the following description of the Cuba-LIF model:

$$i_t^l = \alpha \odot i_{t-1}^l + W_i s_t^{l-1} + U_i s_{t-1}^l + b_i \tag{4.3}$$

$$v_t^l = \beta \odot v_{t-1}^l + (1-\beta) \odot i_t^l - v_{th} s_{t-1}^l \tag{4.4}$$

$$s_t^l = H[v_t^l - v_{th}] \tag{4.5}$$

Gated Recurrent Networks

RNNs learn temporal dependencies by keeping some of the information from previous timesteps using the recurrent connections. However, their training can be unstable due to vanishing and exploding gradient problems, which can prevent the learning of long-term dependencies (Bengio et al., 1994). Gated RNNs, such as LSTM and GRU, can mitigate these problems. Indeed, the gating mechanism allows to better control the flow of information over the timesteps and can create temporal shortcuts which prevent gradient vanishing. For instance, the GRU was proposed in Cho et al., 2014 as a simplification of the LSTM (Hochreiter et al., 1997), with two gates instead of three:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{4.6}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{4.7}$$

$$c_t = tanh(W_c x_t + U_c(r_t \odot h_{t-1}) + b_c) \tag{4.8}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot c_t \tag{4.9}$$

r_t and z_t are the reset and update gates respectively, which are computed with a *sigmoid* activation function. c_t is the candidate state, computed using a hyperbolic tangent (*tanh*) activation function. The reset gate allows to reset the candidate state based on current input only and the update gate controls how much of the previous hidden state will be kept in the current hidden state.

Some gated SNNs inspired by the LSTM model have been proposed (Shrestha et al., 2017; Lotfi Rezaabad et al., 2020; Ponghiran et al., 2021). In Shrestha et al., 2017, a LSTM is converted to a spiking version by using piece-wise linear functions for the activation functions of the gates. An implementation is proposed on the TrueNorth chip (Merolla et al., 2014). A spiking LSTM model that can be directly trained with backpropagation through time is proposed in Lotfi Rezaabad et al., 2020, using spiking activation functions for the gates. In both cases, it must be noted that not only the output of the cell unit is spiking, but also the gates are spiking. A hybrid analog and spiking LSTM is demonstrated in Ponghiran et al., 2021, using spiking blocks as approximation of the gates. This hybrid network benefits from event-based spike accumulation, but at the expense of decomposing each LSTM timestep into 128 SNN timesteps. Moreover, the hidden states are in full precision, and inputs to the spiking blocks must be converted to spikes at each LSTM timestep. In addition, the LSTM model is computationally expensive due to the use of three gates per unit, which highly increases the number of synaptic operations per layer compared to a simple RNN. The GRU and its variants demonstrate that it is possible to achieve similar accuracy with fewer gates per unit (Cho et al., 2014; Ravanelli et al., 2018).

4.2.2 SpikGRU: a Spiking Gated Recurrent Unit

We investigate the benefits of gated units in recurrent SNNs by proposing a new model: SpikGRU (Spiking Gated Recurrent Unit). It is inspired by the current-based approach of the Cuba-LIF and the gated approach of the Light-GRU from Ravanelli et al., 2018, a light version of the GRU model with a single gate. Indeed, SpikGRU can be seen as an extension of the Cuba-LIF model with an additional gate, z, instead of the parameter β. z is computed using the incoming spikes and another set of parameters, W_z, U_z and b_z, and uses a *sigmoid* activation function. The purpose of z is to determine the best combination of the previous state v^l_{t-1} and the input current (or candidate state) i^l_t used in the computation of v^l_t, similar to the update gate in the Light-GRU. We define SpikGRU as:

$$i^l_t = \alpha \odot i^l_{t-1} + W_i s^{l-1}_t + U_i s^l_{t-1} + b_i \tag{4.10}$$

$$z^l_t = \sigma(W_z s^{l-1}_t + U_z s^l_{t-1} + b_z) \tag{4.11}$$

$$v^l_t = z^l_t \odot v^l_{t-1} + (1 - z^l_t) \odot i^l_t - v_{th} s^l_{t-1} \tag{4.12}$$

$$s^l_t = H[v^l_t - v_{th}] \tag{4.13}$$

Fig. 4.1 illustrates the comparison between the LIF, Cuba-LIF and SpikGRU models. The novelty of our approach is that, unlike other spiking versions of gated networks (Shrestha et al., 2017; Lotfi Rezaabad et al., 2020; Ponghiran et al., 2021), we do not discretize the output of the neuronal variables (i, z, v), but rather only the output activations of the recurrent unit are spiking. Thus, as shown in Fig. 4.1, the synaptic connections (N to N) are spiking, while the neuronal operations (N times 1 to 1) are

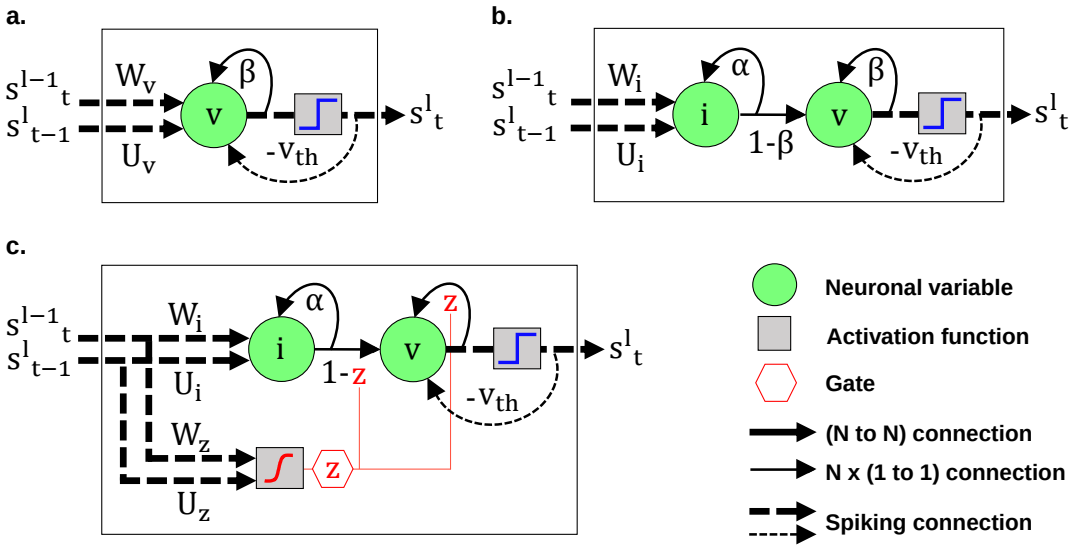


FIGURE 4.1: Recurrent SNN models considered (a. LIF, b. Cuba-LIF, c. SpikGRU), assuming a layer with input and output size **N** and omitting biases for clarity.

kept in full precision. This is similar to the idea of the Cuba-LIF; where v and i are kept in full precision, and v is directly computed from i, introducing more element-wise multiplications (instead of additions). We believe that this increases the accuracy (as removing the discretization of the information on these variables) at the expense of only a small increase in energy consumption. Indeed, these operations occur only at the neuron level (as opposed to synapse level), and hence have a small impact on the energy efficiency (as shown in Chapter 3). In addition, contrary to LSTM networks, we use a unique gate in our model (instead of three), to increase its energy efficiency. Therefore, the number of parameters of SpikGRU is similar to Light-GRU (with one gate). It is approximately 2x higher than the number of parameters of the LIF, Cuba-LIF and Adapt-LIF (which have a similar number of parameters than a RNN), while GRU has 3x times the number of parameters of a RNN (as shown in Table 4.1). Note that a LSTM, with three gates, has 4x more parameters than a simple RNN. Besides, our SNNs learn time constants per neuron, which are additional parameters compared to ANNs. However, these additional parameters are negligible, in particular in the case of fully connected topology (such as RNNs), where the number of neurons is negligible compared to the number of weights.

4.2.3 Experiments on Audio Spiking Datasets

Methods

Datasets and pre-processing. In these experiments, three spiking datasets are used with a classification task to benchmark the SNN models with different degrees of task complexity. **DASDIGITS** (Anumula et al., 2018a) corresponds to the recording from a DAS (64 channels) of the TIDIGITS audio dataset (Leonard et al., 1993). DASDIGITS consists of 11 classes corresponding to the English digits "one" to "nine" plus "oh" and "zero", spoken by 111 (resp. 109) individuals for training (resp. testing) samples. The single digit version of the dataset contains 2,464 training and 2,486 testing samples. The dataset from the CochleaAMS1b sensor is used with a constant time bin

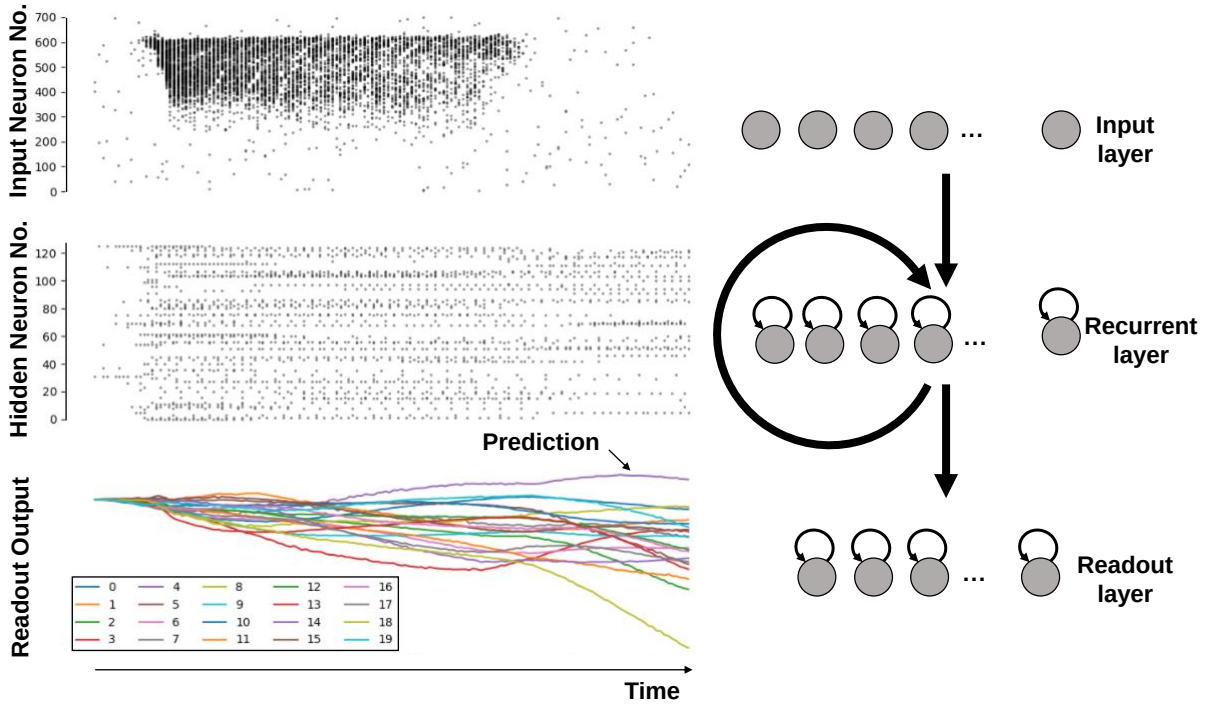


FIGURE 4.2: Sample from the SHD dataset and response from a SNN with one recurrent layer.

pre-processing at 200 Hz. Samples are cut after 1.25 s (almost no spikes are emitted from the sensor after that time) to obtain samples of length 250 timesteps. Therefore, at each timestep, the spike count (number of spikes produced during the time bin) from each channel is fed to both SNN and ANN models in order to compare them with the same data pre-processing. **SHD** and **SSC** datasets (Cramer et al., 2020) are created with an audio-to-spiking conversion procedure inspired by neurophysiology using 700 channels. SHD is a spiking version of the Heidelberg Digits audio dataset consisting in 20 classes of spoken digits in English and German from 12 speakers. It contains 8,156 training and 2,264 testing samples. The test set contains samples from 2 individuals that are not used in the training set plus 5% of samples from other speakers. SSC is a more difficult task based on the Google Speech Command dataset (Warden, 2018). It contains 35 classes corresponding to 35 English words (digits, single word commands and auxiliary words). Samples from 1864 individuals are randomly split between training (75,466), validation (9,981) and test (20,382) sets. SHD and SSC samples have 1s duration and spikes are binned at 250 Hz. The obtained spike count is also fed directly to the models at each of the 250 timesteps.

Training procedure. Neural network topologies with one or two recurrent layers of 128 units and a readout layer (fully-connected to the last recurrent layer), as shown in Fig. 4.2. The readout layer consists of neurons integrating inputs with a self-recurrence, similar to LIF neurons, without the spiking and resetting mechanisms. This readout layer is used for all models (except the ANN GRU) as we empirically observed that it increases the accuracy compared to a standard FC layer. For the training with DAS-DIGITS and SHD, 20% and 10% (resp.) of the training set is used as validation set. To avoid overfitting on the SHD and SSC datasets, noise is introduced in the input

samples during training using spike jitter across channels, as in Cramer et al., 2020; Perez-Nieves et al., 2021. A max-over-time loss (as described in Cramer et al., 2020) is used on the outputs, corresponding to a cross-entropy loss applied on the maximum value of the neurons of the readout layer over all timesteps. All models (SNNs and ANNs) are trained with BPTT, using the approach of the surrogate gradient (Neftci et al., 2019) for the SNNs. In particular, the triangular function (with height 1 and base 2) is used as surrogate for the derivative of the spiking activation function, as in Rathi et al., 2021b. We had also considered using the derivative of the fast sigmoid as a surrogate (as in Zenke et al., 2018). However, we found no effect on the accuracy, as long as the surrogate is not too wide or too narrow (as discussed in Section 2.3). Therefore, the triangular function was used as surrogate gradient for all the experiments presented in this thesis. All weights and biases are initialized from a uniform distribution $U(-k^{-1/2}, k^{-1/2})$, with k being the input size of the layer. The time constants α and β are learnable parameters per neuron and initialized at 0.9. During training, they are clipped between 0 and 1 to avoid unstable behaviors. The spiking threshold v_{th} is set to 1. The input currents i and membrane potentials v are clipped during training as we empirically observed that it improves the model accuracy. Adam optimizer (Kingma et al., 2014) is used with a learning rate 0.001 for 200 epochs and a batch size 128 (512 for SSC). Note that the standard RNN model (ANN) leads to unstable training and low accuracy on these tasks. We mitigated theses problems by initializing the recurrent weight matrices with the identity matrix scaled by a factor (0.5) and using the *ReLU* activation function, similar to Le et al., 2015.

Accuracy on DASDIGITS, SHD and SSC

Table 4.1 shows the average accuracy of the SNN and ANN models on the three datasets with the 1x128 and 2x128 recurrent topologies. We compare our results with previous works on recurrent SNNs on these datasets (except for DASDIGITS, for which we are not aware of other works using similar settings).

For all three tasks, the GRU achieves the best accuracy, except with the 2-layer topology for SSC and SHD where it is similar to the RNN and Cuba-LIF, respectively. However, these tasks may be too easy for the GRU. Indeed, the accuracy is not significantly increased from the 1-layer to the 2-layer topology for DASDIGITS and SHD, compared with spiking models. Moreover, for the SSC task, the GRU shows a high level of overfitting, which is not entirely solved by the addition of spike jitter accross input channels. We observe that the RNN has similar accuracy than the GRU on the DASDIGITS and SSC tasks. However, this RNN does not reach a satisfactory average accuracy on the SHD task, partly due to an unstable training, as shown by the large confidence interval. It is interesting to note that spiking RNNs (LIF and Cuba-LIF) do not present such training instability. This may be due to the self recurrence of spiking neurons that is weighted by a time constant with value close to (but lower than) 1, which may help preventing gradient vanishing.

Comparing SNN models, we observe that the accuracy of the LIF is lower than that of Cuba-LIF on all tasks, up to a 8.4% difference on the SSC task with the 1-layer topology. The 2-layer Cuba-LIF yields 85.5% accuracy on DASDIGITS, which is $<$1% below the accuracy of the 1-layer and 2-layer GRU. Notably, on SHD, the 2-layer Cuba-LIF achieves 87.8% accuracy, which is superior to the accuracy of the 1-layer and

TABLE 4.1: Testing accuracy (%) of the spiking (LIF, Cuba-LIF, SpikGRU) and non-spiking (RNN, GRU) models on the DASDIGITS, SHD and SSC datasets, shown with the 95% confidence interval. The best accuracy for each topology for spiking and non-spiking models is highlighted. Results from related works are also indicated. The number of parameters (#Params) is given for SHD.

	DASDIGITS	SHD	SSC	#Params
	1x128 network			
GRU	**85.9** $\pm$ 1.4	**86.8** $\pm$ 1.2	**75.5** $\pm$ 0.2	321k
RNN	85.8 $\pm$ 1.4	74.9 $\pm$ 3.1	75.3 $\pm$ 0.7	109k
LIF	78.3 $\pm$ 1.9	80.6 $\pm$ 2.0	63.1 $\pm$ 0.8	109k
Cuba-LIF	81.1 $\pm$ 1.1	**83.7** $\pm$ 1.3	71.5 $\pm$ 0.4	109k
SpikGRU	**81.8** $\pm$ 1.1	**83.7** $\pm$ 1.5	**74.7**$\pm$ 0.4	215k
Adapt-LIF* *Yin et al., 2020*	-	79.4	-	109k
Cuba-LIF† *Cramer et al., 2020*	-	71.4	50.9	109k
Cuba-LIF† *Perez-Nieves et al., 2021*	-	82.7	60.1	109k
	2x128 network			
GRU	**86.2** $\pm$ 1.3	**87.3** $\pm$ 0.9	77.9 $\pm$ 0.3	420k
RNN	84.9 $\pm$ 1.4	75.0 $\pm$ 7.3	**78.1** $\pm$ 0.3	142k
LIF	82.7 $\pm$ 0.8	85.8 $\pm$ 1.7	70.3 $\pm$ 1.3	142k
Cuba-LIF	**85.5** $\pm$ 0.9	**87.8** $\pm$ 1.1	75.7 $\pm$ 0.2	142k
SpikGRU	83.3 $\pm$ 1.7	86.4 $\pm$ 1.8	**77.0** $\pm$ 0.4	281k
Adapt-LIF* *Yin et al., 2020*	-	84.4	-	142k
Adapt-LIF Yin et al., 2021	-	87.8	74.2‡	142k

$*$ binary inputs. † 2000Hz pre-processing. ‡ 2x400 network.

2-layer GRU (86.8% and 87.3% resp.). For the more difficult SSC task, SpikGRU outperforms other spiking models for both topologies. Indeed, SpikGRU achieves 74.7% (resp. 77.0%) accuracy with 1-layer (resp. 2-layer) topology, which is only 0.8% (resp. 1.1%) below the best ANN accuracy.

In addition, all the SNNs in our experiments show higher accuracy on the SHD task than the Adapt-LIF from Yin et al., 2020, for the same topology and number of timesteps. However, they use strictly binary inputs, meaning that, if there is more than one spike in the time bin, it is considered as if there were only one (the other spikes are discarded). On the other hand, we directly used the spike count. Indeed, the average input sparsity measured on the testset is only increased from 4.6% to 4.7% (resp. 4.7% to 4.8%) spikes per neuron per timestep on SHD (resp. SSC) for a pre-processing at 250 Hz. Therefore, the additional energy consumption is small while the model accuracy is increased as no spikes are lost. Note that, on SHD and SSC, for a pre-processing with high frequency (such as 2000 Hz), spike count and binary inputs are equivalent as there is never more than one spike per time bin. Our Cuba-LIF also achieves better accuracy than the Cuba-LIF from Cramer et al., 2020; Perez-Nieves et al., 2021 on both the SHD and SSC datasets for the same topology. However, in Cramer et al., 2020; Perez-Nieves et al., 2021, the pre-processing is set at 2000 Hz which results in 2000 timesteps. The higher the number of timesteps, the higher the precision of the inputs,

but also the higher the difficulty of the task. Indeed, it increases the sequence length, making it harder for recurrent units to retain relevant information. Furthermore, the lower accuracy of the Cuba-LIF in Cramer et al., 2020 can be explained by the fact that they use fixed time constants per layer (according to Perez-Nieves et al., 2021). In addition, the best results among the previous works with SNNs on SHD and SSC datasets are demonstrated in Yin et al., 2021, also using a 250 Hz pre-processing. For the same topology their Adapt-LIF shows the same accuracy (87.8%) as our Cuba-LIF on SHD. However, in the SSC task, even with a larger topology (2x400), the accuracy of their Adapt-LIF (74.2%) is lower than the accuracy of our 2-layer Cuba-LIF (75.7%) and SpikGRU (77.0%).

4.2.4 Number of operations in Spiking vs. Artificial RNNs

In the interests of comparing the different neuron models, we have first used the number of MAC and AC operations as a figure of merit for energy efficiency. This allows to have a hardware-independent metric for energy-efficiency, before making assumptions on the hardware implementation. In order to stay as close as possible to the reality, the MAC and AC operations were not translated into their respective energy consumption, as most of the energy consumption of neural networks in specialized architectures comes from memory accesses associated with arithmetic operations rather than from the arithmetic operations themselves (Horowitz, 2014). Furthermore, a detailed comparison of the energy efficiency of RNNs and SRNNs considering different ANN implementations, as done in Chapter 3, is provided Section 4.4.

In these experiments, spiking models exhibit a high sparsity. On the given tasks, SRNNs produce on average between 0.06 and 0.21 spikes per neuron per timestep for processing one sample. The 2-layer Cuba-LIF yields 0.06 spikes per neuron per timestep on DASDIGITS and SSC, which means that a neuron produces on average only 15 spikes during the 250 timesteps. Similarly, the 2-layer SpikGRU achieves 0.09 spikes per neuron per timestep on SSC.

Due to the high spike sparsity, the number of operations per sample is highly reduced compared to an ANN where operations are performed at each timestep. Table 4.2 indicates the number of MAC and AC operations per timestep of one layer of the ANN and SNN models to process a sample. We observe that in ANN models (GRU and RNN) there are mainly MAC operations (except for the bias of neurons), while in SNN there are mainly AC operations (and some element-wise multiplications due to neuronal operations). In SNN models, the number of AC is weighted by the activity rate (spikes per neuron per timestep) of the SNN layers, which decreases (resp. increases) the number of operations if it is inferior (resp. superior) to 1, compared to an equivalent ANN. Note that the Cuba-LIF has similar number of operations than the LIF. Indeed, the input current variable represents only additional MACs at the neuron level, which is negligible compared to the operations in the feedforward and recurrent synaptic connections. On the other hand, the SpikGRU model increases significantly the number of operations compared to LIF and Cuba-LIF due to the additional feedforward and recurrent synaptic connections of the gate.

Fig. 4.3 shows the accuracy vs. total effective number of operations (MAC + AC) per timestep of SNN and ANN models on the three datasets. The number of operations in the 2-layer Cuba-LIF is decreased by 16x compared to the 1-layer GRU while the

TABLE 4.2: Number of MAC and AC operations per timestep for one layer of the ANN and SNN models. m and n are respectively input and output size of the layer. For SNN models, a_{in} and a_{out} are respectively input and output activity rate (spikes per neuron per timestep) of the layer.

Model	Nb MAC	Nb AC
GRU	$3mn + 3n^2 + 3n$	$3n$
RNN	$mn + n^2$	n
LIF	n	$mn * a_{in} + (n^2 + n) * a_{out} + n$
Cuba-LIF	$3n$	$mn * a_{in} + (n^2 + n) * a_{out} + n$
SpikGRU	$3n$	$2mn * a_{in} + (2n^2 + n) * a_{out} + 2n$

models have similar accuracy on DASDIGITS. On SHD, the 2-layer Cuba-LIF even slightly outperforms the 1-layer and 2-layer GRU while reducing by 37x and 49x (resp.) the number of operations. On SSC, the number of operations in the 2-layer SpikGRU is reduced by 8x (resp. 24x) compared to the 2-layer RNN (resp. GRU) while its accuracy is only $\approx$1% below. Compared to the Cuba-LIF on SSC, the SpikGRU model shows better accuracy but at the expense of 2x the number of operations. Our models are compared with the Adapt-LIF from Yin et al., 2021 using the number of MAC and AC operations provided in their paper. Our most accurate 2-layer spiking models are more energy-efficient than the Adapt-LIF. Indeed, the number of operations per timestep is 8.6k (Cuba-LIF) vs. 11.5k (Adapt-LIF) for the SHD task, and 17.6k (SpikGRU) vs. 28.5k (Adapt-LIF) for SSC.

4.2.5 Discussion

Our experiments on the DASDIGITS, SHD and SSC datasets demonstrate the ability of recurrent SNNs to perform classification on sequential data with high sparsity, and hence a very low number of operations. The number of operations in the Cuba-LIF and proposed SpikGRU models is reduced by up to 49x and 24x (resp.) compared to the GRU, for almost the same accuracy ($<$1.1% below). This suggests that having matching timesteps between ANNs and SNNs can result in a higher reduction in the number of operations (*relative to the number of operations of the equivalent ANN*), compared to decomposing each ANN timestep in several SNN timesteps, as it is commonly done in FC and CNN topologies for processing images (in that case ANNs have only 1 timesteps). Indeed, the number of spikes fired is likely to increase with the number of timesteps. This can explain why the sparsity *per ANN operation* obtained here with the SRNNs (between 0.06 and 0.21 spikes per neuron per timestep) are higher than that of the SCNNs studied in Chapter 3. This higher sparsity could also be influenced by the nature of the data, which could lead to a higher network sparsity, and should be further investigated.

Moreover, we demonstrate that the Cuba-LIF model outperforms the LIF model, as it achieves better accuracy for approximately the same number of operations. In

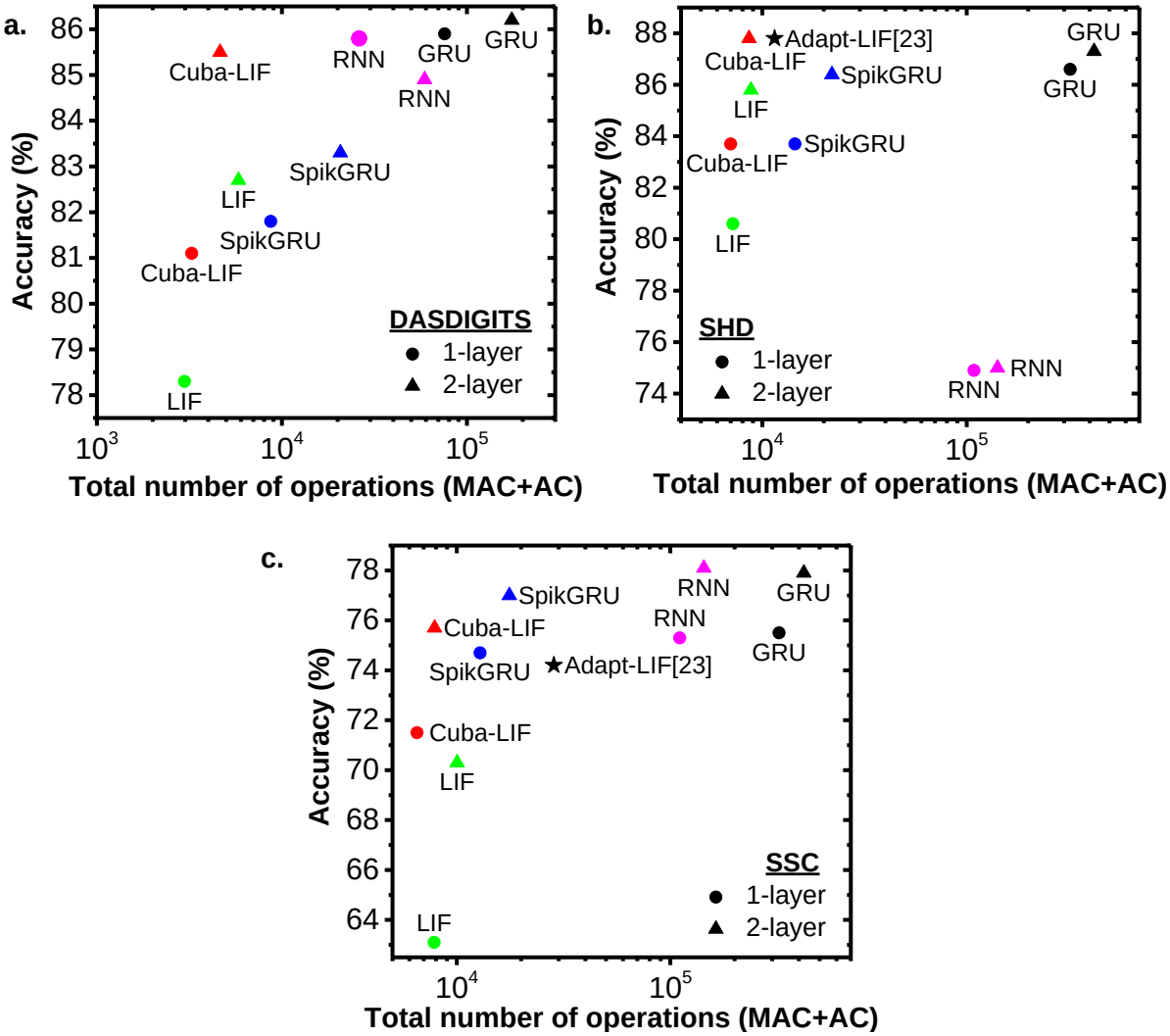


FIGURE 4.3: Accuracy vs. total number of operations (MAC + AC) per timestep for processing one sample from the (a) DASDIGITS, (b) SHD and (c) SSC datasets.

addition, the Cuba-LIF may also outperform the Adapt-LIF model for these tasks. Indeed, the Cuba-LIF achieved better accuracy than the Adapt-LIF from previous works, for a similar model complexity. Furthermore, our proposed SpikGRU model shows a high potential to outperform non-gated recurrent SNNs on more difficult tasks, at the expense of an increased number of operations. However, this must be further investigated. Indeed, we studied tasks with different degrees of difficulty, due to the input size and number of classes, but its ability to retain longer-term dependencies than the Cuba-LIF using tasks with different temporal sequence length should be investigated. Besides, a model of a gated recurrent SNN with a single gate and using real-valued neuronal variables was also proposed in Ponghiran et al., 2022, published after these experiments had been carried out. The main difference is that, in Ponghiran et al., 2022, the hidden state (keeping the long short-term information) is dissociated with the spiking and reset mechanisms. The comparison of the performance of the two models should be further investigated.

Finally, we have observed that SRNNs already yield a high spike sparsity without using any strategy to enhance the spiking activity in these experiments. Therefore,

methods to boost sparsity in SNNs should further increase their energy efficiency, and will be studied in Section 4.3.

4.3 Leveraging Sparsity in Recurrent SNNs

The high sparsity and accuracy demonstrated by SRNNs on spatio-temporal tasks motivated us to further explore both sparsity and recurrent topologies as a way to improve the energy efficiency of SNNs. Indeed, we believe that sparsity can be further increased by training the SNN with no compromise on accuracy. Moreover, we evaluate the benefit of leveraging data sparsity in spiking inputs, such as those produced by spiking audio feature extractors or dynamic sensors, on the energy efficiency.

Keyword spotting (KWS) is a relevant applications for energy-efficient algorithms. KWS, which consists in detecting specific keywords in an audio stream comprising speech, has a wide range of applications such as activation of voice assistants, voice control, speech data mining, routing phone calls, etc. (Lopez-Espejo et al., 2022). ANNs have shown impressive performance on these tasks, but their energy consumption limits their use in embedded systems. Indeed, always-on KWS systems for small electronic devices, such as activation of voice assistants, have power and energy constraints. Spiking FCs or CNNs have been demonstrated for KWS using the Google Speech Command Dataset (GSCD) from Warden, 2018 v1 (Blouw et al., 2020a; Blouw et al., 2020b; Wang et al., 2022) and v2 (Pellegrini et al., 2021). However, SRNNs have not yet been proposed for KWS, although RNNs are well suited for spatiotemporal data such as speech. Moreover, GRU models have shown high performance on low-power embedded hardware for KWS (Kim et al., 2022a). Therefore, SpikGRU, the spiking adapted version of GRU introduced in Section 4.2, promises to achieve high accuracy and efficiency on a KWS task.

In this section, we compare the accuracy-efficiency trade-off in SpikGRU and GRU with different topology sizes, using the KWS task of GSCD v2. We show the benefits of exploiting the sparsity in SpikGRU by regularizing the spiking activity during the training. Then, we explore the advantage of leveraging sparsity in the input data by converting the real-valued inputs into spikes.

4.3.1 Experiments on a Keyword Spotting Task

Dataset and Pre-processing

GSCD v2 contains 35 different words of at most 1 second sampled at 16 kHz. The keyword spotting task consists in a 12-class classification problem with 10 keywords (“yes”, “no”, “up”, “down”, “left”, “right”, “on”, “off”, “stop”, “go”), “silence” (background noise) and “unknown” (non-keyword words) classes. The dataset is provided with 84,843 training, 9,981 validation and 4,890 test samples.The pipeline used in these experiments is shown in Fig. 4.4. From the audio signals, 40 log-Mel features were extracted, with frequencies between 80Hz and 8kHz, window size of 30 ms and hop length of 10 ms. This results in 100 simulation timesteps with the values of the 40 channels being fed to SNNs and ANNs at each timestep. The input samples are re-scaled such that each channel has a unit variance across the time dimension. We used data

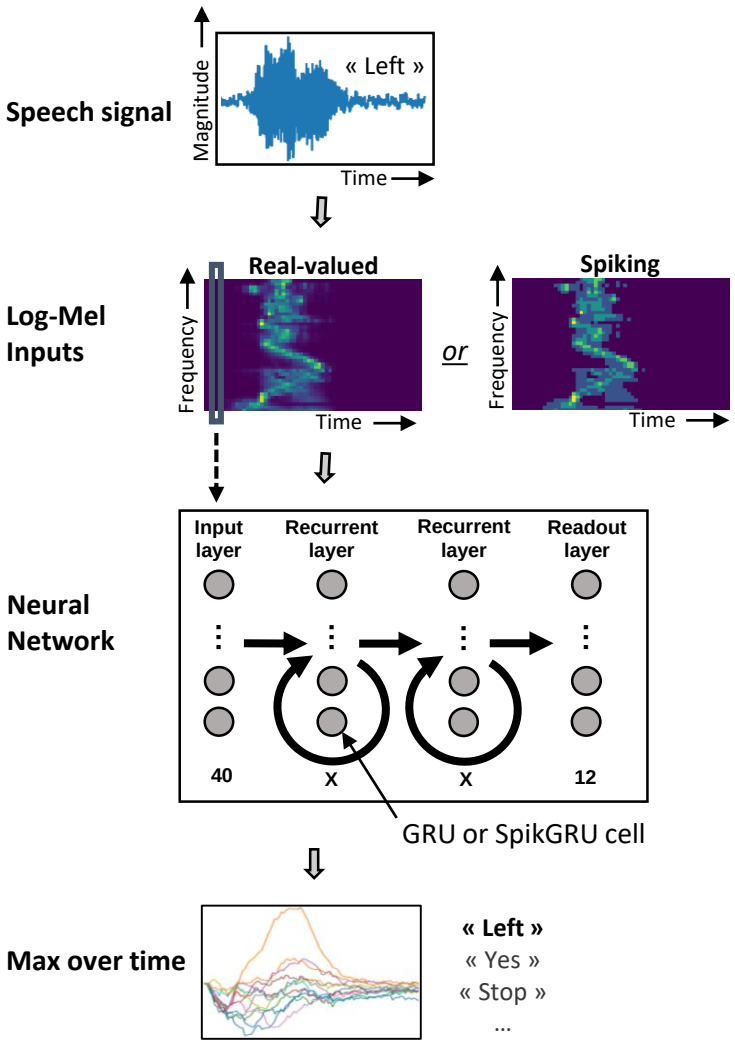


FIGURE 4.4: Keyword spotting task. Log-Mel features are extracted from the raw audio signal and either the real values (experiment 1, Section 4.3.2) or the converted spiking inputs with spike count (experiment 2, Section 4.3.3) are fed to the neural network at each timestep. The neural network has two recurrent layers of X GRU or SpikGRU cells. The maximum value over time of the readout neurons is used for the prediction.

augmentation with background noise and time shift as in Zhang et al., 2017, and time and frequency masking (Park et al., 2019).

In the first experiment (results in Section 4.3.2), the real-valued (32-bit float) log-Mel are directly fed to neural networks. This increases the accuracy of the SNN, as it allows to keep the precision of the input data in the first layer, which results to be very important for the final accuracy of the network Deng et al., 2020. However, this "encoding layer" requires MAC operations. Therefore, in the second experiment (results in Section 4.3.3), the log-Mel features are converted to spikes. For the spike conversion, each 32-bit float value (from the log-Mel spectrogram) is multiplied by a factor K and then rounded to the nearest integer. Four different values of K are chosen, leading to four different input activity rates (i.e. average channel activity per timestep). We have chosen K such as the resulting input activity rates are below 1, such that the number of operations is reduced compared to using an encoding layer. Therefore, by

choosing $K \in \{0.5, 1.0, 1.5, 2.0\}$, the resulting average input activity rate (measured over the test set) $\in \{0.23, 0.48, 0.74, 0.99\}$ spikes per channel per timestep, respectively. Note that channels are allowed to produce more than one spike per timestep (up to $N \in \{5, 11, 16, 21\}$, respectively).

Neural Networks and Training Procedure

The SpikGRU model (described in Section 4.2) is used as SNN baseline and GRU (Cho et al., 2014) as ANN baseline. ANNs and SNNs are composed of two recurrent layers with $X \in \{32, 64, 128, 256, 512\}$ units each, and a readout layer (FC to the last recurrent layer) which is composed of leaky integrators in the case of SNNs. Indeed, we empirically found that, for the same number of parameters and activity, two layers yield better results than one but the improvement was less important for three layers. A max-over-time loss (cross-entropy loss applied on the maximum value of the neurons of the readout layer over all timesteps) is used as in Section 4.2. All models are trained with BPTT, using the surrogate gradient approach for SpikGRU (as in Section 4.2). The models are trained for 100 epochs and a batch size of 128, using Adam optimizer (Kingma et al., 2014) with a learning rate starting from 0.001 and decaying to 0, with a cosine annealing scheduler. For the SNNs, weights and biases are initialized from a uniform distribution $U(-k^{-1/2}, k^{-1/2})$, k being the input size of the layer. The time constant α is a learnable parameter per neuron and initialized at 0.8 and v_{th} is set to 1. In all the experiments, each configuration is run 5 times and the mean accuracy with the 95% confidence interval is reported.

In these experiments (as in Section 4.2), we have used the total number of operations (MACs + ACs) as a hardware-independent figure of merit for energy efficiency. The number of MACs and ACs per layer are computed using the input and output size of the layer, and the input and output spike activity rate (number of spikes per timestep per neuron) for SNNs, in the same way as described in the previous experiments (see Section 4.2).

4.3.2 Increasing Sparsity with Gradient Descent

In this first experiment, we investigated the effect of regularizing the spiking activity during the training of SNNs to decrease the number of spikes and thus the number of operations. For this purpose, we added a term in the loss function to penalize spike firing per layer as in Pellegrini et al., 2021:

$$loss^{l}_{reg} = \frac{1}{2}\frac{1}{N}\frac{1}{T}\sum_{t}\sum_{n} S_n[t]^2 \tag{4.14}$$

with N the number of neurons in layer l and T the number of timesteps per inference. $S_n[t]$ is equals to 1 if neuron n fired a spike at time t, and 0 otherwise. Therefore, the gradient descent algorithm used to train the SNN will minimize both the loss related to the accuracy of the network on the task and the loss related to the spike activity. The regularization loss on the spike activity is weighted by a coefficient λ ($\in \{0.5, 1, 2, 4, 10, 50\}$ in this experiment) allowing us to adjust the impact of the penalization and thus the spiking activity in the network.

In Fig. 4.5, SpikGRU and GRU are compared on the GSCD v2 using the real-valued log-Mel inputs with different network sizes. Regularizing the spiking activity in SpikGRU during training allows us to adjust the trade-off between accuracy and number of operations, as shown in Fig. 4.5 (A). We found that a small λ ($\in \{0.5, 1, 2\}$, depending on the topology) reduces the number of operations with no compromise on accuracy. For each topology, the level of activity regularization leading to the highest accuracy is used for the results in Table 4.3 and Fig. 4.5 (C). In particular, the activity reached using the regularization strategy is on average (for both layers) 0.126, 0.078, 0.049, 0.045 spikes per neuron per inference for X=64, 128, 256, 512 (respectively) compared to 0.191, 0.140, 0.103, 0.071 (respectively) without regularization. This shows that SpikGRU achieves high sparsity, confirming the results of the Section 4.2 with another dataset, with non-spiking data. Besides, it can be noted that the sparsity increases with the size of the topology.

Therefore, SpikGRU with a small activity regularization achieves high accuracy (less than 0.5% below the ANN with similar accuracy) while demonstrating up to 82% reduction in number of operations, as shown in Table 4.3 and Fig. 4.5. Note that this is achieved by using larger topologies than the ANN, but the number of operations is still lower due to the spike sparsity. Moreover, SpikGRU with activity regularization achieves higher accuracy than the previous state-of-the-art SNN (94.5%, demonstrated by a spiking 2D CNN in Pellegrini et al., 2021), while requiring fewer operations (from -39% to -90%, depending on the topology). Interestingly, the SCNN from Pellegrini et al., 2021, also trained with activity regularization, is also very sparse (0.045 spikes per neuron per timestep on average). Indeed, it also uses matching SNN and ANN timesteps, as the 2D convolutions are first performed on the data seen as an image, but the resulting feature maps are re-decomposed in the temporal dimension and fed to the spiking neurons timestep per timestep (using the spectrogram timesteps), as in a RNN. This suggests that, as mentioned in Section 4.2, the higher sparsity is achieved by matching ANN and SNN timesteps, which is allowed by a RNN-like processing in the temporal dimension of the data. However, the CNN topology in Pellegrini et al., 2021 has a higher number of neurons than the RNN topologies from our experiments, resulting in a higher total number of operations, despite a similar sparsity per neuron.

Spikes allows MACs to be replaced by ACs in the synaptic operations of spiking layers. However, due to the use of real-valued inputs instead of spiking ones, an important number of MACs remain in the feedforward connections of the first layer of SpikGRU (encoding layer). These MACs limit the efficiency of SNNs as they are not affected by the activity regularization. In particular, for small topologies with low activity rate, these operations become dominant (Fig. 4.6). Therefore, in the next experiment, we evaluate the impact of using spiking instead of real-valued inputs.

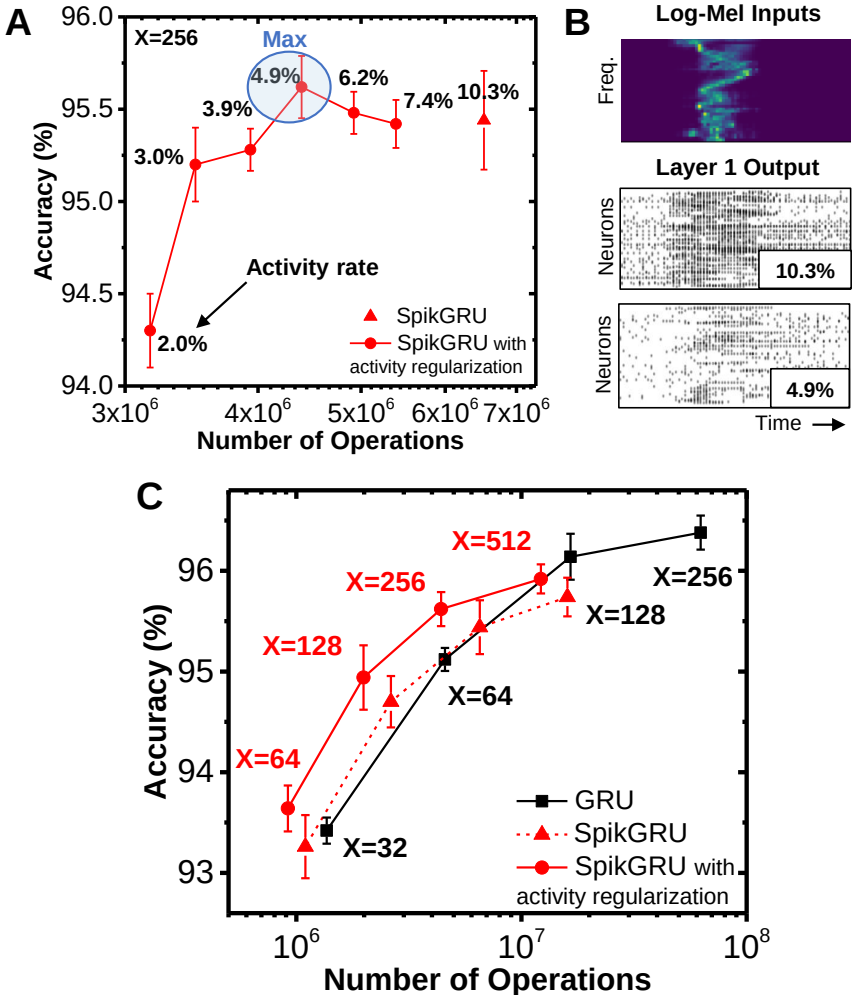


FIGURE 4.5: **A.** Accuracy vs. number of operations per sample for SpikGRU (with size $X = 256$) with the different levels of activity regularization ($\lambda \in \{0.5, 1, 2, 4, 10, 50\}$). **B.** Output spikes from the first layer in SpikGRU with different activity rates. **C.** Accuracy vs. total number of operations (MAC + AC) per sample for GRU and SpikGRU with different layer sizes (X), with and without activity regularization for SpikGRU.

TABLE 4.3: Accuracy and number of operations per sample on GSCD v2 for SpikGRU (with activity regularization) and GRU, and previous state-of-the-art SNNs.

	Topo. (#Param.)	Accuracy (%)	#Ops
GRU	X=32 (14k)	93.4 ± 0.1	1.4M
	X=64 (46k)	95.1 ± 0.1	4.6M
	X=128 (165k)	96.1 ± 0.2	17M
	X=256 (625k)	96.4 ± 0.2	63M
SpikGRU	X=64 (31k)	93.6 ± 0.2	0.9M
	X=128 (111k)	94.9 ± 0.3	2.0M
	X=256 (418k)	95.6 ± 0.2	4.4M
	X=512 (1.6M)	95.9 ± 0.1	12M
SNN SoA*	CNN (130k)	94.5	20M

*from Pellegrini et al., 2021

4.3.3 Increasing Sparsity with Spiking Input Data

In this second experiment, the KWS pipeline is used with the real-valued inputs converted to spikes. Note that activity regularization is also leveraged, using the λ leading to the maximal accuracy for each topology in the first experiment (described in Section 4.3.2). In this case, the input sparsity also impacts the accuracy and efficiency in SpikGRU. Note that the input activity rate must be less than 1 to reduce the number of operations compared to using real-valued inputs (for which 1 MAC operation is performed at each timestep).

The results are shown in Fig. 4.6. The spiking inputs effectively allow to replace the MACs due to the real-valued inputs by ACs (Fig. 4.6 A). However, the reduction in number of operations is limited, as the input activity rate must be high enough to allow the input to be encoded with sufficient precision (otherwise the accuracy is largely degraded). Indeed, we observe that decreasing the input activity rate decreases the accuracy (Fig. 4.6 B). Therefore, the reduction in operations due to spiking inputs is not sufficient to compensate for the loss in accuracy. This leads to a degraded trade-off between accuracy and number of operations compared to the case of real-valued inputs (Fig. 4.6 C).

This experiment shows that the precision of the input is crucial for the network accuracy. Therefore, adjusting the level of activity regularization and network size seems more efficient to decrease the number of operations with a minimal loss in accuracy. However, in hardware implementation, inputs are often quantized (and not in 32-bit float precision, as in these experiments) to increase the energy efficiency. Hence, there is also a trade-off between accuracy and energy consumption in the encoding layer. Besides, the hardware implementation of spike conversion from real-valued feature extractors induces an energy overhead. However, audio feature extractors consuming about 100 nW (such as Wang et al., 2021) output data in the form of spikes, making it possible to yield an ultra-low power end-to-end KWS solution. DAS (such as Anumula et al., 2018b) also directly output spikes.

Therefore, SNNs offer two possibilities for the choice of the feature extractor: either using log-Mel features (quantized or not), or using a spiking feature extractor. In the first option, the first layer is trained to do the spike conversion, which allows minimal accuracy loss. The second option may be more advantageous if the extractor is able to produce relevant spiking features (leading the network to learn with a satisfactory accuracy) with sufficiently high sparsity.

4.3.4 Discussion

The results have shown that recurrent SNNs, such as SpikGRU, are a promising solution for energy-efficient and accurate KWS. We have demonstrated the importance of regularizing the spiking activity, which allows SpikGRU to achieve a better trade-off between accuracy and number of operations than GRU. Indeed, for the same accuracy, SpikGRU shows a lower number of operations. Moreover, the number of operations can be reduced by up to 82% compared to GRU with a loss of accuracy of less than 0.5%. Furthermore, we demonstrate state-of-the-art results for SNNs on GSCD v2 (up to 95.9% accuracy) with extremely low spiking activity, and hence low number of operations. Replacing the real-valued inputs by spiking inputs can further reduce the number of operations by exploiting input data sparsity, but the accuracy-efficiency

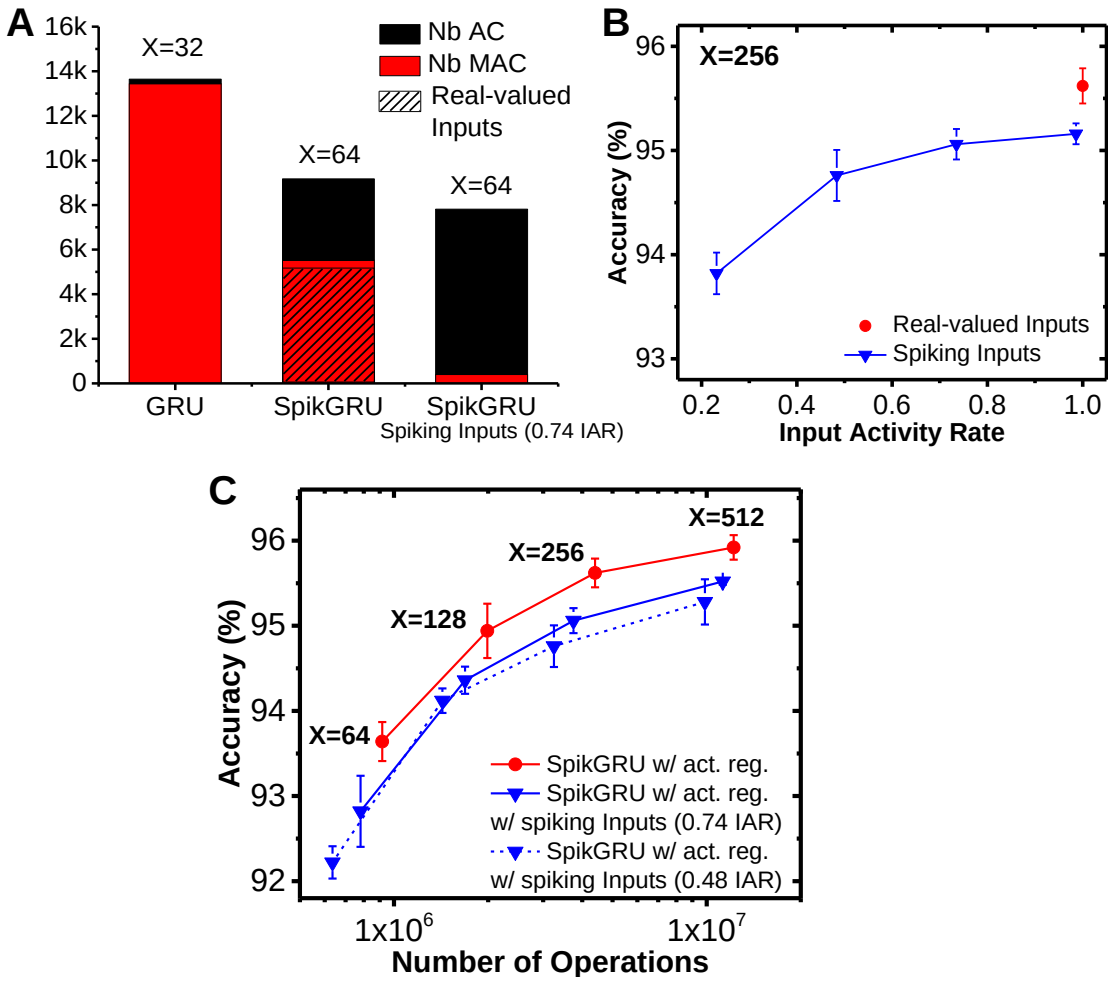


FIGURE 4.6: **A.** Number of MAC and AC operations per timestep for GRU and SpikGRU (real-valued inputs) and SpikGRU with spiking inputs, with Input Activity Rate (IAR) of 0.74. MACs due to real-valued inputs in SpikGRU are highlighted. **B.** Accuracy vs. input activity rate with spiking inputs. **C.** Accuracy vs. number of operations per sample for SpikGRU with activity regularization (act. reg.) with real-valued or spiking inputs (IAR 0.74 and 0.48) with different hidden layer sizes (X).

trade-off must be carefully considered. Therefore, SNNs offer the possibility of using either standard log-Mel feature extractors leading to high accuracy, or spiking feature extractors to increase the energy efficiency. The main limitation of SNNs is the need for larger topologies compared to ANNs with the same accuracy, which increases the memory requirements. This problem could be mitigated by pruning synaptic connections.

Note that the baseline ANN model used in this Section is the standard implementation of the GRU (Cho et al., 2014) using two gates. However, lighter versions of the GRU with only one gate (as in SpikGRU) exists (such as Ravanelli et al., 2018). Therefore, the number of operations of the GRU could likely be reduced by using single gate versions without loosing in accuracy, which should be further investigated. Moreover, a more realistic hardware model must be considered to fairly compare ANN and SNN implementations, which will be the focus of Section 4.4.

4.4 Energy Efficiency of Spiking vs. Artificial RNNs

In Sections 4.2 and 4.3, the energy efficiency of recurrent ANNs and SNNs are compared based on the number of operations. In order to fairly compare them in a more realistic hardware-aware perspective, the energy model from Chapter 3 can be used. However, this model must be adapted to the specificity of RNN topologies (neuron model, data reuse, number of synapses and neurons, etc.). Therefore, this section presents the extension of the energy model desribed in Chapter 3 to the case of recurrent gated topologies. The model SpikGRU (proposed in Section 4.2) with sparsity optimization (Section 4.3) is used as SNN baseline. The ANN equivalent of SpikGRU, i.e. a single-gate GRU, is used as ANN baseline. As in Chapter 3, a baseline ANN model of energy (naive implementation) and ideal cases (with data reuse and exploitation of sparsity) are considered. Then, the KWS experiment (from Section 4.3) is used for the numerical applications.

4.4.1 Extension of the Model of Energy Efficiency to RNN topologies

Following the procedure from Chapter 3, we derive the dynamic energy consumption of each of the models considering the number of operations (MACs and ACs) and memory accesses (as previously, activation functions are ignored). As *LIF+inst* or *IF+const* SNN models in the case of CNN topologies, the SNN models for recurrent topologies require some neuronal operations (e.g. membrane potential decay) that must be realized synchronously at each timestep. In addition, due to the recurrent connections, spikes cannot be processed immediately as they are produced, and must be buffered in between timesteps in an event stack (e.g. a FIFO). Note that the spike sparsity can still be advantageously leveraged compared to an ANN processing where activations are stored in a matrix format. In addition, as the synaptic operations are the dominant source of energy consumption in neural networks, the contribution of these periodic neuronal operations is small, as will be seen.

For the following equations, a recurrent layer with N_{neur_in} (resp. N_{neur_hid}) input (resp. hidden) neurons is considered with a number of input (resp. hidden) synapses N_{syn_in} (resp. N_{syn_hid}) corresponding to feedforward (resp. recurrent) connections. The input (resp. hidden) spiking activity per timestep is $N_{spikes\_in/syn}$ (resp. $N_{spikes\_hid/syn}$). The energy of read (resp. write) the variable X is ER_X (resp. EW_X). The energy per inference (e.g. processing of one sample) is considered, using T timesteps per inference.

SNN Model (SpikGRU)

The dynamic energy consumption per inference of a SpikGRU layer (see equations 4.10-4.13) can be computed. First, the current I and gate variable Z are computed by accumulating the input spikes from the input layer (for the feedforward connections) and the hidden layer (for the recurrent connections). These synaptic operations are realized asynchronously during the timestep. This incurs, at each incoming spike (either from the feedforward or from the recurrent connections), for I and for Z, a memory read and write, the read of the associated weights and an accumulate operation. These synaptic operations are summarized as follows:

$$N_{syn\_in} \times N_{spikes/syn\_in} \times T \times (ER_I + ER_Z + ER_{weight\_in\_I} + ER_{weight\_in\_Z} + 2AC + EW_I + EW_Z)$$
$$+N_{syn\_hid} \times N_{spikes/syn\_hid} \times T \times (ER_I + ER_Z + ER_{weight\_hid\_I} + ER_{weight\_hid\_Z} + 2AC + EW_I + EW_Z)$$

Then, at the end of the timestep, the synchronous neuronal operations are performed. First, the computation of the final version of I (requiring a MAC for the decay, an AC for the bias parameter, a memory read and write) and the final version of Z (requiring an AC for the bias parameter and a memory read only, as Z is not stored in between timesteps). Then the membrane potential V is updated, requiring 2 MACs for the dot products, a memory read and write, and an AC weighted by the spiking activity of the hidden neurons for applying the reset:

$$N_{neur\_hid} \times T \times (ER_I + EW_I + ER_Z + ER_V + EW_V + 3MAC + 2AC + N_{spikes/syn\_hid} \times AC)$$

Finally, the spikes produced are buffered in between timesteps. This results in a memory read for the incoming spikes (from feedforward or recurrent connections) and a memory write for the output spikes produced:

$$(N_{neur\_in} \times N_{spikes/syn\_in} + N_{neur\_hid} \times N_{spikes/syn\_hid}) \times T \times ER_{spike} + N_{neur\_hid} \times N_{spikes/syn\_hid} \times T \times EW_{spike}$$

Therefore, the total dynamic energy consumption per inference of a SpikGRU layer can be summarized as follows:

$$\begin{aligned} E_{SNN} = {} & N_{syn\_in} \times N_{spikes/syn\_in} \times T \times (ER_I + ER_Z + ER_{weight\_in\_I} \\ & + ER_{weight\_in\_Z} + 2AC + EW_I + EW_Z) \\ & + N_{syn\_hid} \times N_{spikes/syn\_hid} \times T \times (ER_I + ER_Z + ER_{weight\_hid\_I} \\ & + ER_{weight\_hid\_Z} + 2AC + EW_I + EW_Z) \\ & + N_{neur\_in} \times N_{spikes/syn\_in} \times T \times ER_{spike} \\ & + N_{neur\_hid} \times T \times (ER_I + ER_Z + ER_V + EW_I + EW_V + 3MAC + 2AC \\ & + N_{spikes/syn\_hid} \times (AC + ER_{spike} + EW_{spike})) \end{aligned} \tag{4.15}$$

ANN Model: Baseline

In the KWS experiments, the standard ANN GRU implementation (with two gates) was used. However, in order to fairly compare ANN and SNN only considering their processing mode, the same topology must be used. Therefore, **a light version of GRU (with a single gate) is considered to be the ANN equivalent of SpikGRU**, in terms of model complexity. It is defined as follows (similar to Ravanelli et al., 2018):

$$c_t = ReLU(W_c x_t + U_c h_{t-1} + b_c) \tag{4.16}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{4.17}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot c_t \tag{4.18}$$

The synaptic operations are first computed to update the neuronal variables C and Z. In this case, the input activation (*iact*), C, Z and their associated weights must be read, 2 MACs operations (one for each) are performed and C and Z are updated. Note that, in SpikGRU equation 4.15, the contribution of feedforward and recurrent synapses is separated to show that the associated spiking activity must be used for each. In the equations for the ANN, the two contributions are combined for clarity. Hence, N_{syn} is the total number of synapses in the layer and *iact* represents any of the inputs, either from the feedforward (x_t) or recurrent (h_{t-1}) connections. These synaptic operations are summarized as follows:

$$N_{syn} \times T \times (ER_{iact} + ER_C + ER_Z + ER_{weightC} + ER_{weightZ} + 2MAC + EW_C + EW_Z)$$

Then, the final state of C and Z is obtained by adding the biases (2 ACs) and computing the activation functions. The update of neuronal variables C and Z costs an additional memory read only (as they are not stored in between timesteps) for each of them, i.e. one per hidden neuron. Then, H is updated, requiring a memory read and write for the update and 2 MACs for the dot products. These neuronal operations are summarized as follows:

$$N_{neur\_hid} \times T \times (ER_C + ER_Z + ER_H + EW_H + 2MAC + 2AC)$$

Therefore, the total dynamic energy consumption per inference of the ANN layer (naive implementation) can be summarized as follows:

$$\begin{aligned} E_{ANN(naive)} = {} & N_{neur\_hid} \times T \times (ER_C + ER_Z + ER_H + EW_H + 2MAC + 2AC) \\ & + N_{syn} \times T \times (ER_{iact} + ER_C + ER_Z + ER_{weightC} + ER_{weightZ} \\ & + 2MAC + EW_C + EW_Z) \end{aligned} \tag{4.19}$$

ANN Model: Ideal Data Reuse

As in Chapter 3, the cases of ideal data reuse and exploitation of *iacts* sparsity are considered. Compared to the naive implementation, the neuronal operations are unchanged, these optimizations targeting the synaptic operations.

In the case of recurrent layers (as in fully connected layers), there is no possible reuse on weights (as each weight is used in only one computation). However, an *iact* can be reused for all hidden neurons, and a *psum* (here the neuronal variables Z and C) can be reused for all of its input neurons. For instance, this could be realized in systolic arrays by associating each *psum* to a PE and moving the input activations through the PEs (output stationary dataflow, according to Sze et al., 2017). In this case, the *iact* is read in the buffer once for each input neuron. Similarly, the *psums* (C and Z) are

read and written in the buffer once for each hidden neuron. Thus, the distant memory accesses for *iacts*, C and Z are reduced to:

$$N_{neur\_in} \times T \times ER_{iact} + N_{neur\_hid} \times T \times (ER_C + ER_Z + EW_C + EW_Z)$$

Then, these variables are accessed locally in a register file (ER^{reg}, EW^{reg}) for each synaptic operation, the corresponding weights are accessed from the distant memory (no reuse on weights), and the MACs are performed:

$$\begin{aligned} N_{syn} \times T \times (ER_{weightC} + ER_{weightZ} + 2MAC + ER^{reg}_{iact} + ER^{reg}_{C} + ER^{reg}_{Z} \\ + EW^{reg}_{C} + EW^{reg}_{Z}) \end{aligned}$$

Including the neuronal operations (which are the same as in the baseline case, see equation 4.19), the total dynamic energy consumption considering ideal data reuse is :

$$\begin{aligned} E_{ANN(reuse)} = {} & N_{neur\_hid} \times T \times (ER_C + ER_Z + ER_H + EW_H + 2MAC + 2AC) \\ & + N_{neur\_in} \times T \times ER_{iact} + N_{neur\_hid} \times T \times (ER_C + ER_Z + EW_C + EW_Z) \\ & + N_{syn} \times T \times (ER_{weightC} + ER_{weightZ} + 2MAC + ER^{reg}_{iact} + ER^{reg}_{C} + ER^{reg}_{Z} \\ & + EW^{reg}_{C} + EW^{reg}_{Z}) \end{aligned} \tag{4.20}$$

ANN Model: Ideal Data Reuse and Exploitation of *Iacts* Sparsity

Then, the ideal exploitation of sparsity in the *iacts*, in addition to an ideal data reuse, is considered. As in the case of CNNs (described in Chapter 3), when an *iact* is zero, MAC operations and memory accesses associated with weights, *psums* and *iacts* can be saved. Therefore, the difference with the case of data reuse only (equation 4.20) is that the synaptic operations (factor N_{syn}) are weighted by the input activity. Note that contributions of feedforward and recurrent connections should be distinguished as they may have a different sparsity, as in the case of SNN. Here, for clarity, they are not dissociated, and γ is supposed to be the average sparsity (rate of zero *iacts*) of both types of connections. Thus, the total dynamic energy consumption in the case of ideal data reuse and exploitation of *iacts* sparsity is:

$$\begin{aligned} E_{ANN(reuse+sparsity)} = {} & N_{neur\_hid} \times T \times (ER_C + ER_Z + ER_H + EW_H + 2MAC + 2AC) \\ & + N_{neur\_in} \times T \times ER_{iact} + N_{neur\_hid} \times T \times (ER_C + ER_Z + EW_C + EW_Z) \\ & + N_{syn} \times T \times (1-\gamma) \times (ER_{weightC} + ER_{weightZ} + 2MAC \\ & + ER^{reg}_{iact} + ER^{reg}_{C} + ER^{reg}_{Z} + EW^{reg}_{C} + EW^{reg}_{Z}) \end{aligned} \tag{4.21}$$

4.4.2 Comparison of Gated Recurrent ANNs and SNNs

Using these equations, the relative energy efficiency of single-gate recurrent ANNs and SNNs are compared. The topologies from the KWS experiments (Section 4.3) are used for the application. In particular, the neural networks have two recurrent layers

of same size (X). The input is an encoding layer (real-valued inputs) and the readout layer does not produce spike. Thus, only the efficiency of hidden layers is compared. As they have the same size, $N_{neur\_in} = N_{neur\_hid} = X$. In addition, for clarity, the feed-forward and hidden spiking activities are combined in an average $N_{spikes/syn}$. For the case of ideal data reuse and exploitation of *iact* sparsity in ANN, the only missing information for the numerical applications is the *iacts* sparsity of the ANN algorithm. Note that using *tanh* activation functions, which are typically used in GRU implementations (Cho et al., 2014), activations are not naturally sparse. However, using *ReLU* activations instead (as proposed in Ravanelli et al., 2018) can result in higher sparsity. As the average sparsity that could be reached in such ANN encoded in fixed point format for a given topology size is unknown, it is supposed to be 50% ($\gamma = 0.5$) in the following numerical applications. For the energy of memory accesses and computations, the reference of 16 bit in 65nm technology as in Section 3.4 (see Table 3.1) is used.

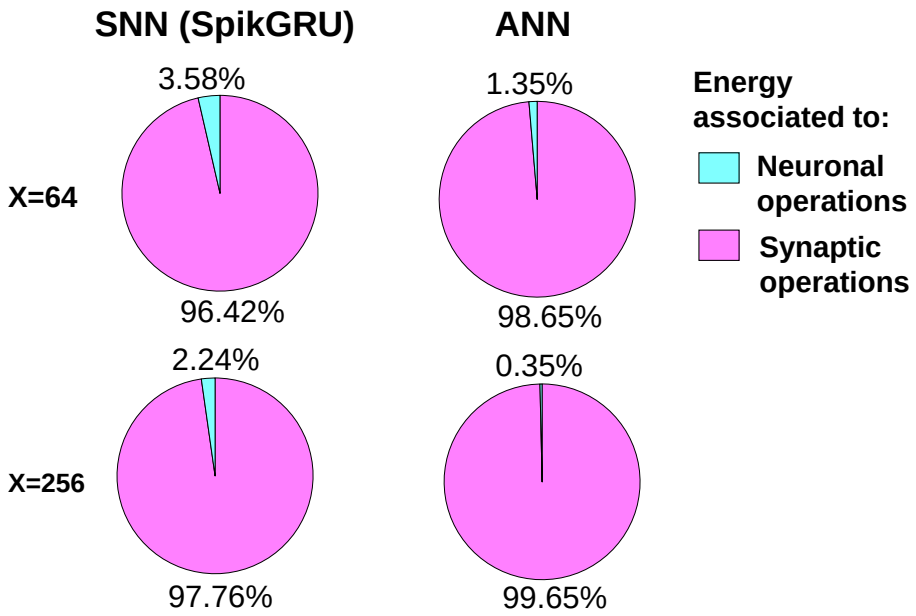


FIGURE 4.7: Ratio of energy associated with neuronal operations and synaptic operations in the SNN and ANN with ideal data reuse and exploitation of *iact* sparsity (equations 4.15 and 4.21), for two topology sizes (X).

The ratio of energy associated with neuronal operations and synaptic operations in the SNN (SpikGRU) and its ANN equivalent (in case of ideal reuse and exploitation of *iacts* sparsity) is shown in Fig. 4.7. As expected, the synaptic operations are largely dominant (as $N_{syn} = N_{neur}^2$). Therefore, in SpikGRU, the energy associated with spike buffering and the update of neuronal variables is almost negligible compared to the synaptic operations. In particular, the impact of having high precision (as opposed to spiking) neuronal variables requiring MAC operations (as opposed to ACs) is negligible, as we had assumed for proposing the SpikGRU model. Conversely, reducing the operations and memory accesses associated with synaptic operations is crucial for efficiency, both in ANNs and SNNs, as it was the case for CNN topologies. This confirms the important impact of spike sparsity in SNNs, as well as data reuse and exploitation of *iacts* sparsity in ANNs.

The SNN energy efficiency relative to ANN with the same topology depending on the ANN implementation is shown in Fig. 4.8. The case of CNN topologies (from

Fig. 3.7) and RNN topologies (equations 4.15-4.21) is compared (topology size X=256 is used in the case of RNN). RNN topologies appear to be more favorable to SNNs (relative to ANNs) than CNN topologies (i.e. $N_{spikes/syn}$ can be higher for achieving the same relative efficiency), except in the baseline case. This is explained by the fact that there is no reuse on weights in fully connected layers (and so in recurrent), while every data types can be reused in convolutional layers. For instance, for the SNN to achieve superior energy efficiency than the ANN in the ideal reuse case, the average activity per timestep must be at most 0.53, while the average activity per inference must be at most 0.28 in the case of CNN. Note that, to compare with ANNs, the activity must be considered per timestep in the case of RNNs while it must be considered per inference (i.e. for processing one sample) in the case of CNNs. Indeed, recurrent ANNs have the same number of timesteps as SNNs, while convolutional ANNs do not have timesteps (only one forward pass per inference).

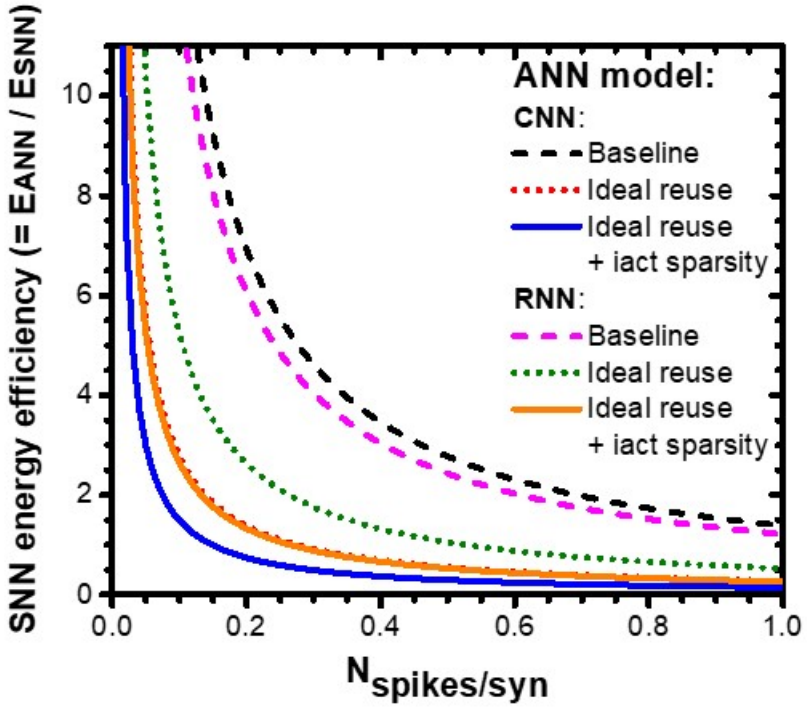


FIGURE 4.8: SNN energy efficiency compared to the ANN (with same topology, with X=256 hidden neurons in recurrent layers) as a function of the SNN spiking activity ($N_{spikes/syn}$ being the average number of spikes per synapses), depending on the ANN implementation. Baseline corresponds to the naive implementation, while Ideal reuse and Ideal reuse + iact sparsity consider ideal ANN implementations, considering only data reuse, or data reuse and the exploitation of *iact* sparsity, respectively. The case of CNN and RNN topologies is compared. The data for CNN are extracted from Chapter 3 while the equations for RNNs are described in this Section.

The impact of different topology sizes on the relative energy efficiency of SNNs vs. ANNs is compared in Fig. 4.9. The experiments on the KWS task described in Section 4.3 are used for numerical applications. For SpikGRU, the case with full-precision inputs and activity regularization is used. In particular, $N_{spikes/syn}$ per timestep was on average 0.135, 0.084, 0.053, 0.048 for X=64, 128, 256, 512 hidden layer size respectively, when using a total of 100 timesteps per inference. First, SNNs and ANNs are

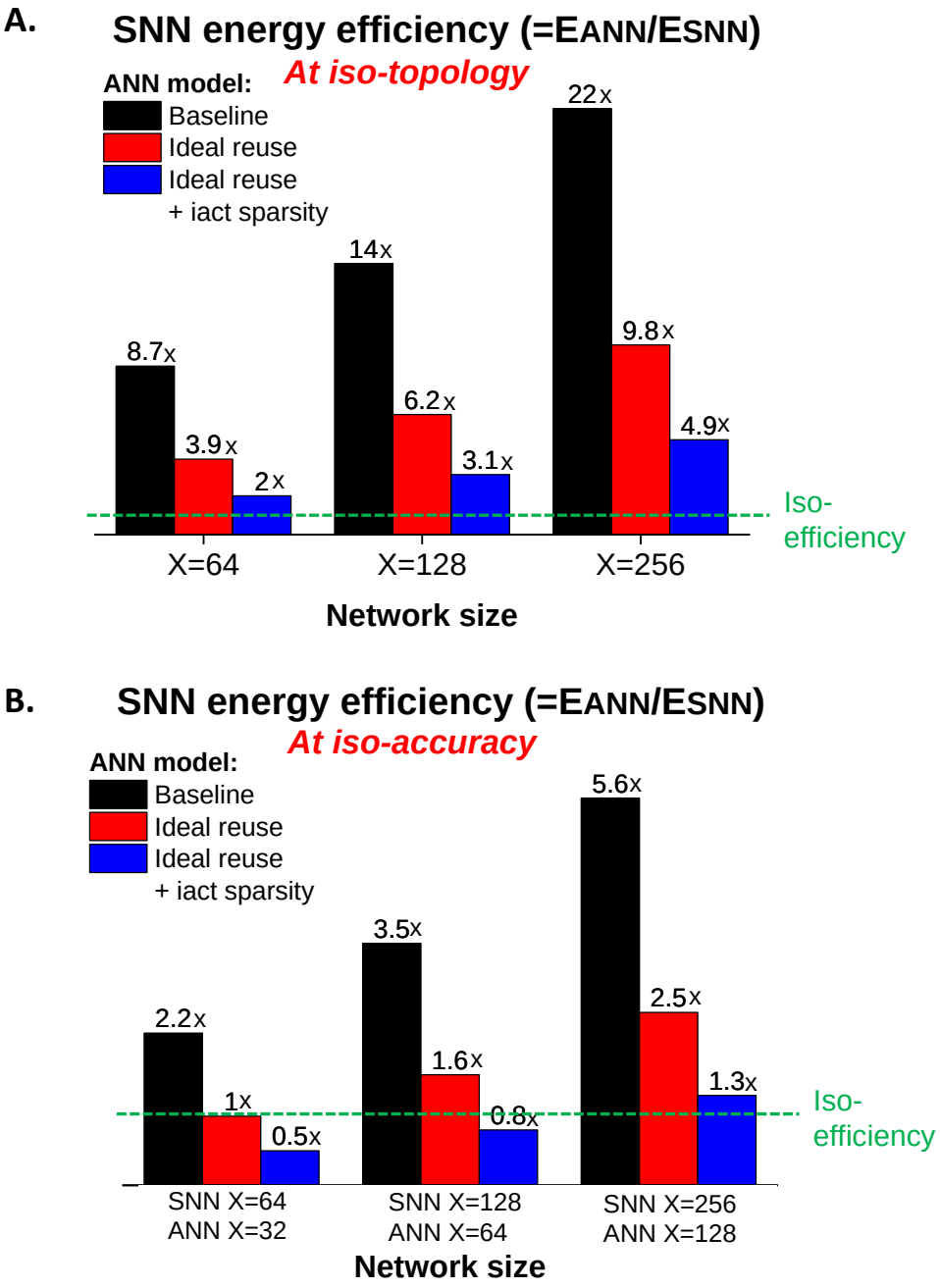


FIGURE 4.9: SNN energy efficiency compared to ANN depending on the ANN implementation for recurrent topologies, in the case of **A.** same topology for ANN and SNN or **B.** similar accuracy for ANN and SNN. For the hardware implementation, SpikGRU is used as SNN model and a light version of the GRU equivalent to SpikGRU (i.e. with a single gate) is used for ANN. Data come from the KWS experiments described in Section 4.3. Note that the accuracy results for the ANN were obtained with the standard GRU implementation (two gates).

compared using the same topology. Second, SNNs and ANNs are compared for a similar accuracy achieved, which approximately corresponds to twice the number of hidden neurons (so four times the number of parameters) for the SNN with respect to the ANN, according to the results on KWS. It is observed that, the bigger the topology,

the higher the relative SNN energy efficiency compared to ANN. This is due to an increasing spike sparsity with the size of the topology. In iso-topology case, the energy efficiency of SNN is always higher than the ANN in baseline and ideal cases (from 2x to 22x more energy-efficient). At iso-accuracy, the energy efficiency of SNN is relatively smaller, as the large network size highly impacts the efficiency. However, the SNN is still more energy-efficient than the ANN (1.3x to 5.6x), except for the ideal reuse + *iact* sparsity case when using small topologies.

4.4.3 Discussion

The gain in energy efficiency of using SNNs instead of ANNs for recurrent topologies is more moderate when considering hardware implementations with higher fidelity than when simply using the number of operations. Indeed, the latter does not consider the opportunities of data reuse and sparsity exploitation in ANNs. However, when comparing the results from Chapter 3 with those obtained here, it is observed that the gain is higher with RNNs than with CNNs. This can be explained by two factors: (1) the higher sparsity in spiking RNNs (likely due to the fact of not having more timesteps in the SNN than in the equivalent ANN), and (2) the lower data reuse opportunities in RNNs (no reuse on weights). Notably, the gains are highly dependent on the SNN sparsity, as well as the choice of topology. Indeed, the wider the hidden layers, the higher the gain for the SNN relatively to the ANN (due to a higher SNN sparsity).

However, the results depend on many assumptions that must be carefully considered when using this model. In particular, assumptions were made on the ANN recurrent model, for instance its sparsity. Similar experiments, as done in the case of SNNs, should be performed with the assumed ANN model (with *ReLU* activations) to evaluate the sparsity for each topology case. Note that the sparsity is likely to increase with the topology size, as in SNNs, while it was considered fixed in this section. Therefore, the conclusions that SNNs compare favorably to ANNs with increasing topology size must be verified for the case of ANN exploiting *iact* sparsity. In addition, as in the case of CNNs, exploiting sparsity requires additional logic consuming energy and makes it more difficult to reuse data (according to Chen et al., 2019). Therefore, a realistic ANN model would likely show lower performance than the ideal cases described above. As done in Chapter 3, the performance of real ANN implementations should be evaluated based on existing accelerators dedicated to RNNs (see Mittal et al., 2021 for a review). In addition, the results at iso-accuracy assume that the SNN needs twice as many hidden neurons as the ANN to achieve similar accuracy, which was according to the results obtained on KWS. However, these results must be verified considering the new assumptions made on the ANN (in particular, a GRU with a single gate and *ReLU* activations, instead of the standard GRU with double gate and *tanh* activations that was used in the KWS experiments).

4.5 Conclusion

Spiking RNNs can achieve accuracy close to ANNs with higher energy efficiency, as shown in the case of spoken word recognition, using various spiking and non-spiking datasets. The proposed SpikGRU model leverages the sparsity of SNNs, due to sparse

spiking activations, and the accuracy of GRU, by using a gated cell model with high precision neuronal variables and operations. By carefully considering the topology size and the activity regularization, the accuracy and energy efficiency of SpikGRU can be adjusted to match the application constraints. It can also enable efficient end-to-end spike processing by combining it with low-power spiking feature extractors.

The extension of our model of dynamic energy consumption (presented in Chapter 3) to the case of recurrent topologies provides a new perspective on the energy efficiency of SNNs and ANNs. First, it confirms that the neuronal operations are negligible compared to synaptic operations, both in ANNs and SNNs, validating the motivations for the SpikGRU model (i.e. with high precision neuronal variables and operations, and spiking activations). Moreover, the impact of spike sparsity on the SNN efficiency, as well as the impact of data reuse and exploitation of *iacts* sparsity on the ANN efficiency, are again demonstrated. In this context, the conclusions on the relative energy efficiency of ANNs and SNNs are different than that of Chapter 3. Indeed, on the one hand, a higher sparsity (per ANN operation) is achieved in spiking RNNs than in spiking CNNs and, on the other hand, there are less opportunities of data reuse in artificial RNNs than in artificial CNNs. Therefore, SNNs seem to compare more favorably to ANNs in the case of recurrent topologies, in particular for wider topologies, which exhibit the highest spike sparsity (and also the highest accuracy). Note that this is contingent to a high level of spike sparsity, and that SNNs may require wider topology than ANNs for the same accuracy, inducing higher memory requirements.

These experiments seem to confirm our hypothesis that spatio-temporal tasks could better match the temporal dynamics of SNNs, compared to static tasks such as image recognition, and hence could lead to higher energy efficiency. This is of high interest for edge AI applications requiring low power and low energy consumption, such as keyword spotting. Furthermore, the benefits of recurrent SNNs, such as SpikGRU, could extend to many applications using sequential data, where RNNs are typically used, such as speech recognition, text summarization (Shini et al., 2021), sentiment analysis (Baktha et al., 2017), but also time series forecasting using environmental (Chen et al., 2018) or physiological (Mao et al., 2022) data. In addition, other spatio-temporal tasks, such as gesture recognition, in particular with data produced by neuromorphic sensors, should be further investigated.

Chapter 5

Improving Accuracy and Efficiency with Analog Synapses

5.1 Introduction

Neuromorphic computing, such as bio-inspired algorithms and hardware architectures, is a promising research direction for improving the energy efficiency of hardware implemented ANNs (Nawrocki et al., 2016; Bose et al., 2019). Among neuromorphic architectures, analog implementations can achieve significant gains compared to fully-digital implementations (such as Hung et al., 2022). In such systems, emerging Non-Volatile Memory (NVMs) devices (Ielmini et al., 2019), such as resistive random-access memories (RRAMs), magnetic RAMs (MRAMs), phase-change RAMs (PCRAMs) or ferroelectric RAMs (FeRAMs), can be used to encode synaptic weights of ANNs (for instance, to replace the SRAMs used in standard fully-digital hardware implementations). The non-volatily of these devices allows them to retain the information even if the power supply is turned off, allowing to remove the dependency to an external memory and hence reducing the energy consumption of the system. In addition, multi-level cell programming strategies allow more than one bit of information to be stored in a single NVM device. Therefore, multi-level NVMs can allow energy-efficient and dense hardware implementations of ANNs (such as Joshi et al., 2020; Wan et al., 2022).

Nevertheless, emerging NVMs are prone to variability, inducing the occurence of errors, which can significantly degrade the accuracy of ANNs. This trend is exacerbated with dense multi-level approaches due to the reduced programming window for each level (Nirschl et al., 2007; Balatti et al., 2013). Therefore, enhancing the robustness of neural networks to noisy multi-level weights is essential to achieve accurate and efficient hardware implementations of ANNs. In addition, variability in NVMs comes from different sources and results in different types of errors, which can have a different impact on the accuracy of ANNs (Higuchi et al., 2022; Yan et al., 2023).

Among NVMs, RRAMs are one of the most promising technologies to implement synaptic weights of neural networks (Yao et al., 2020). Synaptic weights with RRAMs are usually implemented using 1 Transistor 1 Resistor (1T1R) architectures, where the memory (1R) is associated with an access transistor (1T). However, by using 1 Selector 1 Resistor (1S1R) architectures, where the memory is co-integrated in series with a back-end selector (1S), instead of 1T1R, the memory density can be improved by about one order of magnitude (Minguet Lopez et al., 2021). High density memory is particularly important to implement large network topologies.

In this chapter, we simulate ANNs and SNNs with the constraints of a hardware implementation using analog NVMs for the synaptic weights. First, we consider a technology-independent hardware model that is, in principle, applicable to all kind of emerging analog NVMs. This model is used to study the robustness of ANNs and SNNs to errors. This allows to compare the SRNN model proposed in Chapter 4, SpikGRU, to its ANN equivalent, as well as with CNN topologies (spiking and non-spiking), in terms of robustness to errors. Second, a case study is considered, using RRAMs with a 1S1R architecture to implement the synaptic weights of a Binary SNN (BSNN).

This chapter is organized as follows:

- In Section 5.2, the hardware fault model to simulate implementations with NVMs is proposed. Then, a training methodology for neural networks adapted to highly-quantized and noisy weights is presented. The robustness of different neural networks topology (CNN, RNN) and coding (ANN, SNN) is compared.
- In Section 5.3, a practical case of resistive memories is considered to implement a BSNN. Physical measurements performed on a memory array are used to extract the hardware constraints.

The results of this chapter have been published in Minguet Lopez et al., 2021; Minguet Lopez et al., 2022; Dampfhoffer et al., 2023d; Minguet Lopez et al., 2023.

5.2 Improving Robustness to Noisy Quantized Weights

In the interest of implementing synaptic weights of neural networks using emerging NVMs, several challenges must be considered, in particular concerning the variability of NVM devices. The robustness of ANNs to NVM non-idealities has been shown to depend on the topology of ANNs. For instance, wide and shallow neural networks are more robust than deep networks (Yang et al., 2019b). Besides, Spiking Neural Networks (SNNs) are thought to be particularly robust to noise in synaptic weights, due to the computations using accumulation over time (Li et al., 2020). The robustness of ANNs and SNNs to noisy synaptic weights have been compared in Li et al., 2020; Bhattacharjee et al., 2022. However, Li et al., 2020 do not consider a realistic hardware implementation, as weights are simulated with 32-bit floating point precision and only one type of error is considered. A more realistic hardware model of a RRAM memory array is used to evaluate the robustness of SNNs and ANNs in Bhattacharjee et al., 2022. Nevertheless, none of these works (Yang et al., 2019b; Li et al., 2020; Bhattacharjee et al., 2022) consider the benefits of injecting noise during training, which has proven very effective to enhance the fault tolerance of neural networks (Murray et al., 1994). Strategies based on injecting noise during training have been demonstrated with NVM implementations, such as Joshi et al., 2020; Doevenspeck et al., 2021; Minguet Lopez et al., 2022; Wan et al., 2022. However, these works focus on a specific hardware implementation and do not distinguish the effect of the different types of faults on the neural network performance. In addition, to the best of our knowledge, there has been no attempt to evaluate and compare the robustness of CNN and RNN topologies in the context of an implementation with non-ideal NVMs.

Therefore, we propose a methodology to evaluate and improve the robustness of neural networks in the case of noisy multi-level NVM-based synapses. This general methodology is applicable to various types of neural networks and NVM technologies. Moreover, we show that considering the characteristics of NVMs during the training of ANNs is essential to optimize the overall system performance. In particular, we define an abstract hardware fault model distinguishing two types of errors, namely static and dynamic, capturing the variability of NVMs. We compare neural network topologies (CNNs and RNNs) and coding strategies (ANNs and SNNs) on a keyword spotting task. The performance of the neural networks are compared with a standard (error-agnostic) training and with an adapted (error-aware) training. In addition, we propose an analysis of the impact of the error-aware training on the parameters learned by the neural networks. These findings could be used to further improve the performance of neural networks by considering the specificity of multi-level NVM implementations.

5.2.1 Fault Model and Training Strategy for NVM-based Synapses

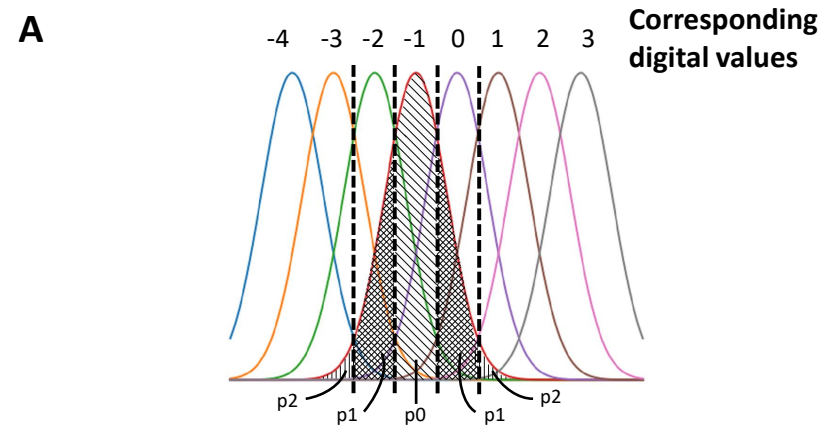
Fault Model for NVM-based Synapses

Single-level (as opposed to multi-level) NVMs are only programmed in two states. Therefore, when one device is used to encode one weight, they allow to implement Binarized Neural Networks (BNNs), with one level encoding positive weights (+1) and the other encoding negative weights (-1). In this case, the Bit Error Rate (BER, i.e. proportion of bit reading failures among all bit read) of the NVMs corresponds to the probability of reading one state instead of the other. Thus, errors can be simply modelled, in the neural network inference, by switching the value of the weights with a given probability.

In the case of multi-level NVMs, the model is more sophisticated. Indeed, the different levels of the NVM do not have the same error rate and the probability for each level to be read as another level must be considered. In addition, multi-level NVMs can be used in different ways to implement ANNs in hardware. The most mature and widely used approach consists in using the NVMs only to store the weights while performing the computation of the matrix vector multiplication in digital. It is the approach used in Near-memory computing (NMC). In-memory computing (IMC) is another strategy that leverages the physical properties of the devices to directly perform the matrix vector multiplication in the memory array (such as Joshi et al., 2020; Amrouch et al., 2021; Jung et al., 2022; Wan et al., 2022). In this study, the NMC approach is modeled, however, the method can be extended to IMC, as will be discussed.

Fig. 5.1 shows the proposed fault model for multi-level NVM-based synapses. In the following experiments, NVMs with eight levels (as in Nirschl et al., 2007; Balatti et al., 2013) are used as an example, but the model can be extended to any number of levels. Moreover, one NVM encodes one weight, which means that the synaptic weights in the neural networks can take only eight different values. Hence, each level is associated with a corresponding digital value that will be used in the digital computation, signed integers ranging from -4 to +3. A gaussian distribution is used to model the variability in the NVM (Grossi et al., 2016). This represents the variability in the analog value that is retrieved (for instance a current or voltage) compared to the expected value. All levels are assumed to have the same variability and to be equally distanced. Therefore, the noise level is determined by the variance of the gaussian distribution.

Note that, for some technologies, the variability depends on the programmed resistance, and thus is different between levels (Grossi et al., 2016). However, by programming the levels such that the overlap in the gaussian distribution of neighboring levels is similar (i.e. by adjusting the distance between levels based on their variability), it is assumed that we could obtain an equivalent NVM model.



B

Nominal p_i (probability of reading a level at distance i)

Noise level	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
1	0.78	0.11	$9*10^{-5}$	$2*10^{-10}$	≈0	≈0	≈0	≈0
2	0.50	0.23	0.02	$4*10^{-4}$	$2*10^{-6}$	$1*10^{-10}$	$1*10^{-13}$	≈0
3	0.32	0.23	0.09	0.02	$2*10^{-3}$	$2*10^{-4}$	$5*10^{-6}$	$1*10^{-9}$

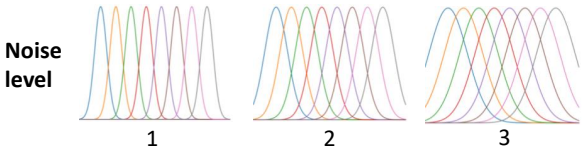


FIGURE 5.1: **Fault model for multi-level NVMs. A.** Noise model in 8-level NVMs (digital values associated to each level in these experiments are indicated). p_i is the probability to read the level at distance i (in particular p_0 is the probability to correctly read the level). **B.** Probabilities of errors (p_i) depending on the distance between two levels, for three noise levels (varying sigma in the gaussian distribution).

As shown in Fig. 5.1, three variances are used to obtain three noise levels. p_i represents the area under the curve of a given level that is between the reading thresholds of a level at distance i. Thus, p_i corresponds to the probability for a given level to be read as the level at distance i, and hence p_0 corresponds to the probability that a given level will be read correctly. Note that the nominal values of p_i given in Fig. 5.1 correspond to the case of a level having an infinite number of neighbors on each side. In practice, the p_i should be adapted for each level depending on the number of levels considered. For instance, in the case of the 8-level NVM shown in Fig. 5.1, considering level "-3", its p_1 on the left side (corresponding to reading "-4") is higher than its p_1 on the right side (reading "-2"). Indeed, there are no other neighbors on the left side, and hence, p_1 on the left side is equal to the remaining area under the curve on the left side of the reading threshold between "-4" and "-3". Moreover, the levels on the extremity ("-4"

and "3" in this case) have a larger p_0 than the other levels, as they only have neighbors on one side.

The abstract error model covering the variability of NVMs is defined with two categories of errors, namely static and dynamic (modeled with the same gaussian distribution). Static (respectively dynamic) errors are defined as fixed (respectively changing) during the inference time. Hence, when testing with static errors, the errors are sampled once for each input data and used for the whole inference. When testing with dynamic errors, an identical and independent sampling of the error distribution is performed each time the weight is read. In practice, these errors result from different physical effects. Depending on the technology, static errors can correspond to programming failures, or temporal fluctuations of the device with a timescale higher than the inference time, such as conductance relaxation for RRAMs (Zhao et al., 2017), or drift for PCRAMs (Karpov et al., 2007). Dynamic errors can correspond to the inherent noisy behavior of analog devices (Minguet Lopez et al., 2022; Yan et al., 2023; Reganaz et al., n.d.). In addition, read operations on FeRAMs are destructive (Ielmini et al., 2019), thus errors are always dynamic as the device is re-programmed after each read. Static and dynamic errors have to be distinguished as they do not have the same impact on neural networks, for example in the case where weights are read several times during the inference. In this experiment, it is assumed that weights are read at each timestep for the RNN and at each incoming spike for the SNNs (assuming an event-based hardware implementation). On the contrary, for CNNs, it is assumed that the architecture only reads the weights once per inference due to data reuse techniques (Sze et al., 2020). Note that static and dynamic errors can exist simultaneously. However, in this study, they are considered separately to understand their respective impact.

Training Neural Networks with NVM-based Synapses

Training with quantized weights. Binary and multi-level NVMs impose high quantization constraints on the weights, if the number of level is limited and only one device is used to encode one weight. Training a neural network with highly-quantized weights poses several challenges and thus requires a specific training procedure. Indeed, the gradient descent algorithm requires high-precision parameters to be able to optimize them smoothly. To circumvent this problem, a full-precision version of the weights is kept during the training (as in Courbariaux et al., 2016). Moreover, with highly-quantized weights, the range of weights is highly constrained, and thus may not match the dynamic required by each layer of the network (e.g. depending on its size). Therefore, in order to allow the network to adjust the range of input values, we use a scaling for each layer which multiplies the activations. The scaling is defined for each layer as a learnable parameter. All operations occuring at the neuron level for processing the input activations to produce the output activations are illustrated in Fig. 5.2. In addition, the learning rate (i.e. factor used in the weight update determining the speed of learning of the parameters) must be carefully set. Indeed, if weights are fixed with values from -4 to +3, these parameters have a higher magnitude than the others. Thus, the learning rate must be scaled so that all parameters learn at the same pace, as proposed in Courbariaux et al., 2016.

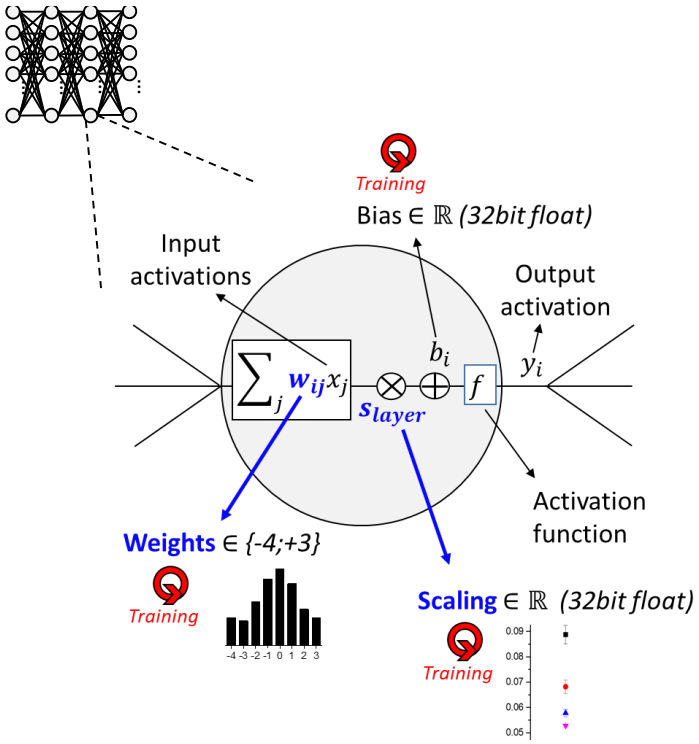


FIGURE 5.2: **Neural Network with highly-quantized weights.** Illustration of the operations for computing the output activation of a neuron. Input activations are multiplied with the quantized weights (8 levels) and by a scaling (full precision, 1 per layer) to adjust the range of the pre-activation. A bias (in full precision, 1 per neuron for RNNs and 1 per channel for CNNs) is added before the activation function. Weights, scaling and bias are trained.

Error-aware training. Single- and multi-level NVMs pose another challenge due to the high noise level, causing errors when reading the weights during inference. However, these errors can be incorporated in the training process, which greatly improves the accuracy of neural networks, as will be seen in the experiments. The proposed general methodology for training with errors is applicable to all kind of neural networks. The incorporation of errors during training will be called “error-aware” training, while the classic way of training will be called “error-agnostic". In the error-aware training condition, errors on the weights are applied during training, both in the forward pass and the backward pass. However, the weight update is performed on the error-free full-precision version of the weights, which are then quantized to obtain the updated quantized weights. The noise level injected during training must be similar to the noise level that is expected during test, in order to obtain the best performance. Note that, if the neural network is trained with a high noise level, the training from scratch may be slow to converge. As a solution, the models can be initialized with the weights of the models trained in the error-agnostic condition in order to speed-up the training.

In the error model previously described, two types of errors, namely static and dynamic, are defined. However, BPTT training with dynamic errors is costly, as it would require to make different copies of the neural network for taking into account the different errors on the weights at each timestep. Therefore, only static errors (which are fixed during the inference) will be used in error-aware training. Moreover, training

with static errors also improves the robustness to dynamic errors, as will be seen in the results. It is important to note that static errors are sampled for each input data. Therefore, the neural network does not learn which particular synapses are faulty, but rather that all synapses are potentially faulty. Thus, this training procedure targets a general robustness to errors rather than a robustness to a specific configuration of errors (Joshi et al., 2020; Burel et al., 2022).

Note that the accuracy could be further increased by re-training the neural network involving the hardware in the loop. For instance, Moon et al., 2019; Wan et al., 2022 propose to fine-tune the neural network using the measured outputs of the fabricated circuit to account for its specific errors. However, this strategy is costly as it must be done after the chip fabrication and repeated for each hardware unit. Conversely, the only knowledge required in the proposed methodology is the expected overall noise level of the devices.

5.2.2 Robustness of Neural Networks to Noisy Weights

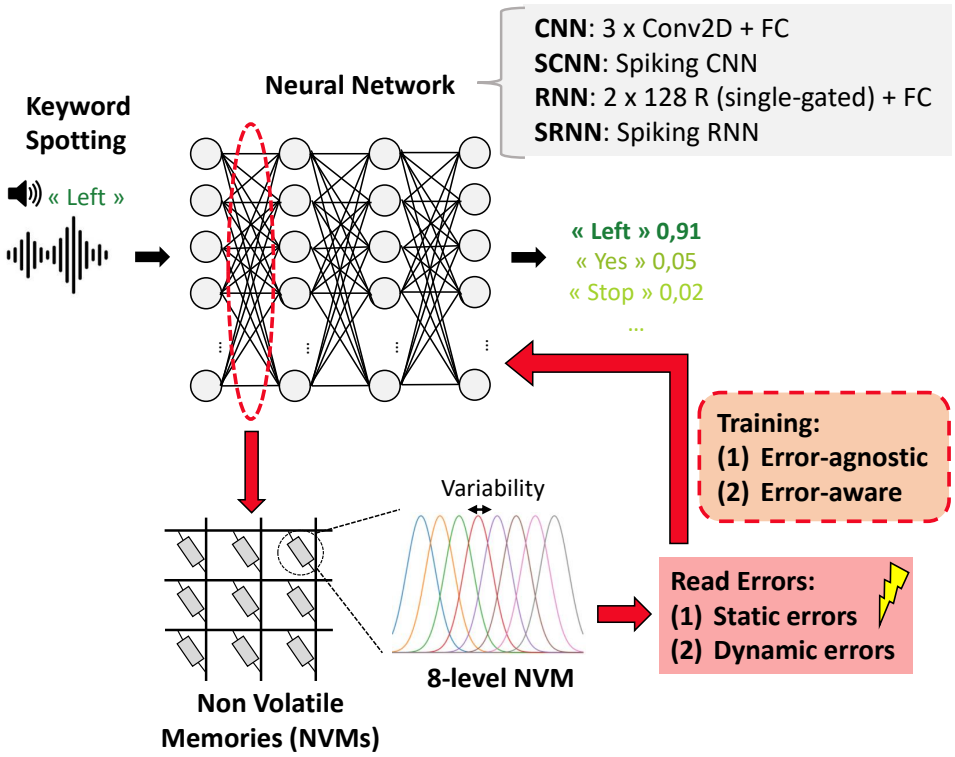


FIGURE 5.3: **Experiments on the keyword spotting task**. Four types of neural networks using 8-level NVMs for weight implementation are simulated on a keyword spotting task. The effects of two training strategies (error-agnostic and error-aware) and two types of errors (static and dynamic) are considered.

Methods

The KWS task from GSCD v2 (Warden, 2018) is used for the experiments. The dataset and data pre-processing are the same as described in Section 4.3. The only difference

is that the log-Mel features are extracted using a hop length of 15 ms (instead of 10ms), resulting in 67 frames (instead of 100), in order to speed up the training.

Four different models are used in the experiments: a Spiking CNN (SCNN), a Spiking RNN (SRNN), and their corresponding ANN equivalent (CNN and RNN), as shown in Fig. 5.3. The CNN and RNN topologies are chosen to have a similar number of parameters and operations per inference (for the ANN versions). For the SNNs, the input data are not converted into spikes but kept in full precision, using an encoding layer (as in Section 4.3 for the non-spiking experiments). This allows to have the same inputs for the ANNs and SNNs, and also to reach high accuracy for the SNNs (as observed in Section 4.3).

The SRNN in these experiments corresponds to the SpikGRU model proposed in Chapter 4. The ANN equivalent of SpikGRU, called RNN in this section, corresponds to the single-gated implementation described in equations 4.16-4.18. Note that the *ReLU* activation function is replaced by a *tanh*, as we have observed that it leads to better robustness to errors. For the RNN and the SRNN, the 40 frequency channels are fed to the neural networks at each timestep, using the same timesteps for the ANN and SNN versions (as in Chapter 4).

The CNNs use 2D convolutions, and hence the input data are processed as images of size *time x frequency*, corresponding to (H,W) dimensions. The CNN topology has three 2D convolutional layers and a final FC layer. The convolutional layers have 16, 32, 64 output channels respectively, with a kernel size of (6,4), a stride of (2,2) and a padding of (2,2). The SCNN is composed of LIF neurons. As the data are processed in a static way (as images), SNN timesteps are added to simulate the SNN temporal dynamics. In these experiments, 10 timesteps were empirically found to be a good compromise between accuracy and spike sparsity. Hence the input images are fed to the SNN 10 times. Note that we used a standard implementation of 2D SCNN, which considers the input as an image and decomposes the inference in "artificial" SNN timesteps, as done in standard image processing. An alternative solution is proposed in Pellegrini et al., 2021, which uses an implementation of a RNN-like SCNN (as explained in Section 4.3.2). However, in these experiments, the objective is to compare the robustness of the CNN-like processing compared to the RNN-like processing.

The neural networks are implemented and trained based on the NVM model and training procedure previously described in Section 5.2.1. Thus, the weights are uniformly quantized with signed integers from -4 to +3. All neural networks are trained with backpropagation for 100 epochs and a batch size of 128, with Adam optimizer (Kingma et al., 2014) and a cosine annealing scheduler. The initial learning rate is set at 0.001, except for the weights, for which it is set at 0.01. Bias parameters (not quantized) are initialized from a uniform distribution $U(-1/\sqrt{k}, 1/\sqrt{k})$, k being the input size of the layer. The scaling parameter per layer is initialized to $1/\sqrt{k}$. The full-precision versions of the weights are initialized from a uniform distribution $U(-1, 1)$. For SNNs, the time constants are defined as learnable parameters (per neuron in SRNN and per channel in SCNN) and initialized at 0.8. The voltage threshold for the spiking activation function is set to 1 and a surrogate gradient is used for BPTT as in Chapter 4. Neural networks are trained with two conditions: error-agnostic or error-aware, as described in Section 5.2.1. In all the experiments, for each configuration, the models are trained five times and the mean accuracy with a 95% confidence interval is reported. Note that, as randomness is involved when testing with errors, models are tested ten

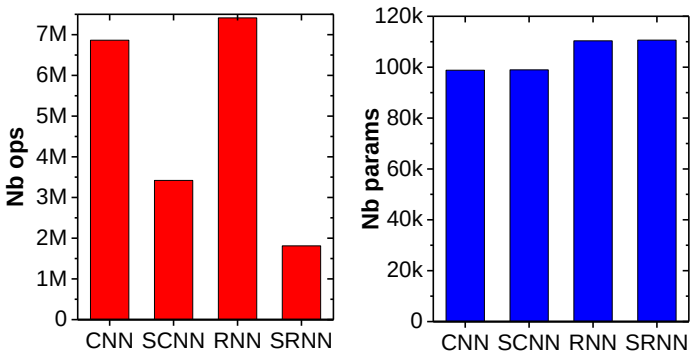


FIGURE 5.4: Number of operations per inference (left) and number of parameters (right) of the models.

times and the mean accuracy is used.

Results

The figures of merit of the different neural networks are provided in Fig. 5.4 and Fig. 5.5. The number of operations per inference and the number of parameters are given in Fig. 5.4. All models have a similar number of parameters. On the one hand, the ANN versions (CNN and RNN) have a similar number of operations per inference. On the other hand, the spiking versions have a lower number of operations due to the spike sparsity. Indeed, the SCNN (respectively SRNN) shows 2x (respectively 4x) reduction in operations compared to the CNN (respectively RNN). Indeed, neurons in SCNN have an average firing rate of 0.54, 0.43, 0.34 spikes per inference for the three convolutional layers respectively. The SRNNs have an average spike activity of 0.18 and 0.12 per timestep for the first and second recurrent layers respectively.

The accuracy of neural networks on the different training conditions (error-agnostic and error-aware) is shown in Fig. 5.5. The two types of errors (static and dynamic) and the different noise levels (described in Section 5.2.1, noise level 0 corresponding to the error-free case) are considered. In the error-agnostic training, the accuracy is largely degraded by errors, with up to 48% accuracy loss in the case of the highest noise level. Nevertheless, error-aware training is very efficient at increasing the robustness of neural networks to noise, even in the worst noise level scenario. However, the higher the noise level, the higher the accuracy gap with the error-free case. Indeed, the accuracy loss with respect to the error-free case is less than 1% for the lowest noise level, but up to 3% for the highest noise level.

Notably, some differences between the neural networks can be observed. First, RNN topologies (SNN and ANN) seem inherently (in error-agnostic training) less robust to static errors than CNN topologies (SNN and ANN). Indeed, the CNN and the SCNN have only 30% and 28% accuracy loss (respectively) in the case of highest noise level, compared to 43% and 48% for the RNN and the SRNN (respectively). This could be explained by the high temporal depth of RNNs, which could lead to errors accumulation and hence accuracy degradation (Yang et al., 2019a). Indeed, although spatially shallow (they have a small number of layers), RNNs are very deep in time. In fact, the output from a recurrent layer is used as input at the next timestep. This means that, when the RNN is unrolled in time, it has an equivalent depth of 67 in the

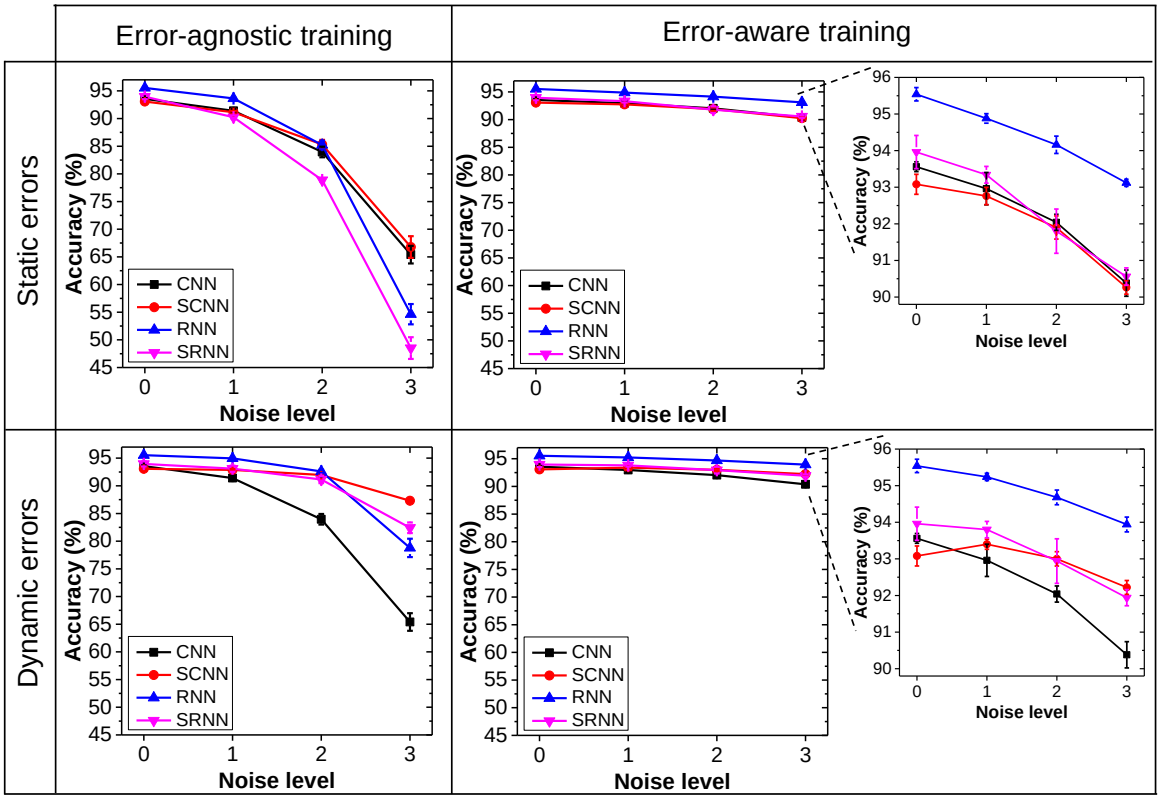


FIGURE 5.5: Test accuracy of the different neural networks (shown with a 95% confidence interval) on the keyword spotting task. Static (top) and dynamic (bottom) errors are considered with different noise levels (cf Fig. 5.1), noise level 0 corresponding to the error-free case. Error-agnostic training (left) and error-aware training (right) are compared. In error-aware training, models are trained at the same noise levels as those used for testing.

temporal dimension (67 being the number of timesteps in these experiments). Nevertheless, the results demonstrate that RNN topologies can recover as much accuracy as CNN topologies with the error-aware training. Note that the SCNN also has a temporal depth due to the use of 10 timesteps to simulate the temporal dynamics of SNN. However the number of timesteps is small compared to the case of the RNN and the SRNN. This is in line with the results in Bhattacharjee et al., 2022, showing that a lower number of timesteps can mitigate the accumulation of errors over time in SNNs.

Second, SNNs and RNNs appear to have a better robustness to dynamic errors than CNNs, in both error-agnostic and error-aware training conditions. This is consistent with the results in Li et al., 2020, where SNNs are found to be more robust than ANNs to dynamic errors in the case of CNN and FC topologies. Indeed, as explained by the authors, synapses in CNNs or FCs are used only once per inference. On the contrary, a synaptic weight in SNNs is used as many times as the number of spikes transmitted across the synapse during inference. In the case of dynamic errors, each weight read for a given synapse is sampled from the same gaussian distribution. Therefore, the more spikes transmitted across the synapse, the more the gaussian noise is minimized. Note that even if the average number of spikes per synapse is lower than 1, some neurons are activated more frequently. Therefore, some synapses transmit a large number of spikes. However, this property does not apply to the case of static errors, where the

same faulty weights are used for the whole inference. In addition, the results show that this property is also applicable to RNNs. Indeed, RNNs have the same behavior as SNNs in the sense that a synapse is reused at each timestep of the inference, if the input activation is not null. Note that CNNs are not sensitive to the difference between static and dynamic error in an ANN implementation, (they are considered identical in this simulation), as explained in Section 5.2.1. For comparison purpose, the CNN was tested with the same implementation as SCNN (i.e. with the weights being read at each pixel). However, even in this case, the results obtained with dynamic errors are similar to the results obtained with static errors. This shows the importance of the accumulation over time in the same synapse, for the dynamic errors to compensate.

Therefore, SNNs, and in particular SpikGRU, seem to be a good choice for an energy-efficient analog implementation, as they benefit from an increased robustness to dynamic errors, as well as a significant reduction in operations per inference.

5.2.3 Analysis of Error-Aware Training

In this section, we provide an analysis of the impact of error-aware training on the parameters learned by the neural networks. As a reminder, in the neuron model with highly-quantized weights, input activations are multiplied by the weights and a scaling factor, which are learned by the neural networks, before the activation function (see Fig. 5.2). In these experiments, it is found that the weights and scaling parameters are highly impacted by the noise level during training, as shown in Fig. 5.6.

Indeed, for almost all layers, the scaling learned in the error-aware training is lower than in the error-agnostic training. Moreover, the higher the noise level during training, the lower the scaling learned. In addition, in the error-agnostic condition, the neural networks learn a gaussian-like weight distribution centered on 0. On the contrary, in the error-aware condition, the weights are distributed more equally among levels, except for the levels on the sides (corresponding to the highest weight magnitude), which are the most used. Moreover, the average magnitude of the weights learned with error-aware training is higher than with error-agnostic training. In addition, as for the scaling, the higher the noise level during training, the more this trend is exacerbated.

Two reasons can explain the change in weight distribution (and therefore the magnitude increase). First, in the error-aware training condition, the weights are distributed more uniformly compared to the error-agnostic training condition, with less weights at level "0". Hence, the neural network may increase its robustness to errors by making more weights specialized, as proposed in Wan et al., 2022. Second, the NVM levels have a different Signal-to-Noise Ratio (SNR), as shown in Fig 5.7. On the one hand, outer levels benefit from a higher weight magnitude (signal) and a lower error magnitude (noise). On the other hand, middle levels have a lower magnitude and a higher error magnitude. Indeed, as shown in Fig 5.1, the outer levels have a higher probability of being correctly read (p_0) compared to the other levels, as they only have neighbors on one side. Moreover, the error magnitude between two levels depends only on the distance between those levels (it is equal to the difference between the two values encoded by these levels), and not on the magnitude of the digital value encoded by this weight. For instance, mistaking a "-4" for a "-3" has likely less impact on the neural network accuracy than mistaking a "0" for a "+1", while these two errors

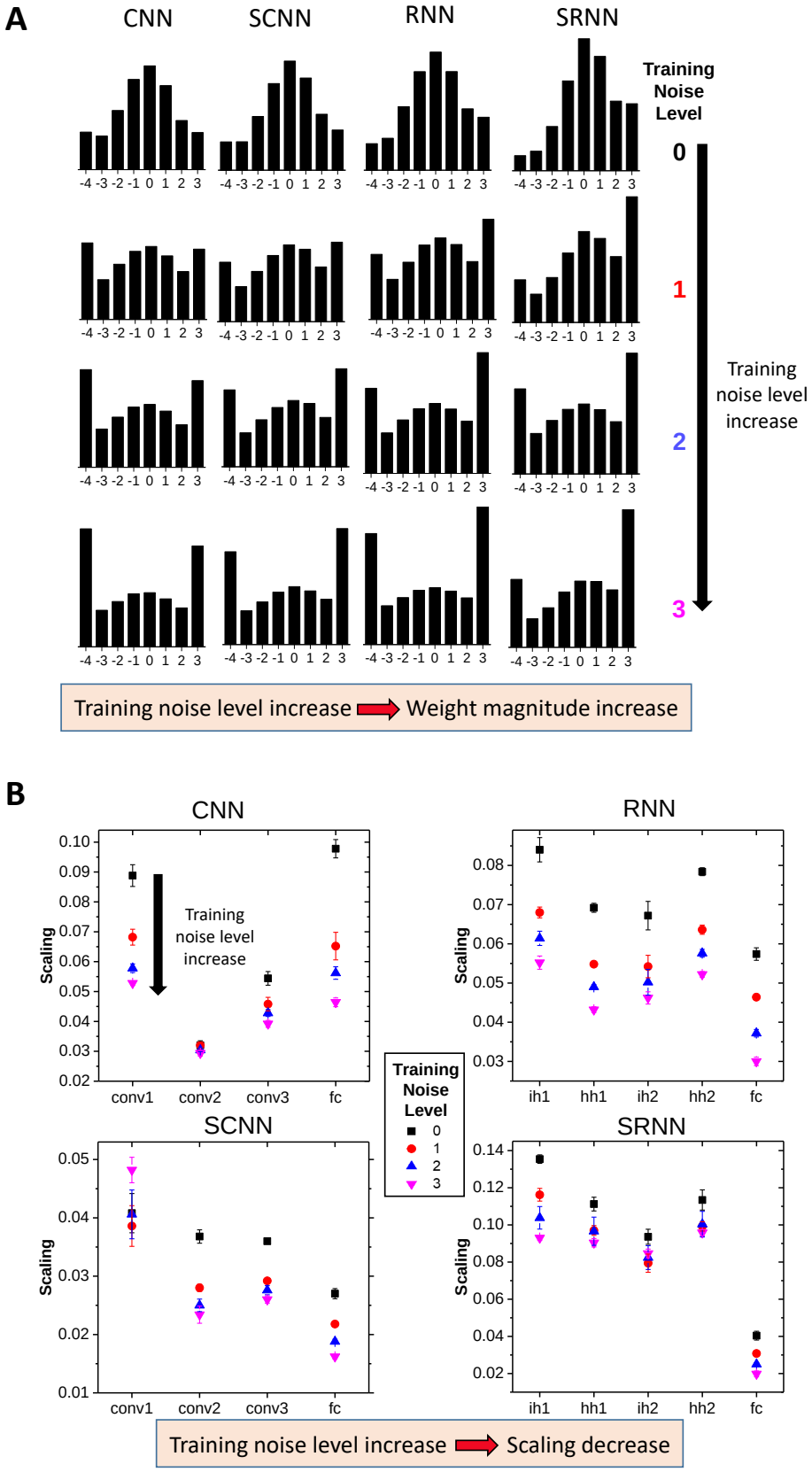


FIGURE 5.6: **Analysis of error training.** Weight distribution (**A**) and scaling of each layer (**B**) for the different models after training, when the models are trained with different noise levels (cf Fig. 5.1), noise level 0 corresponding to the error-free case.

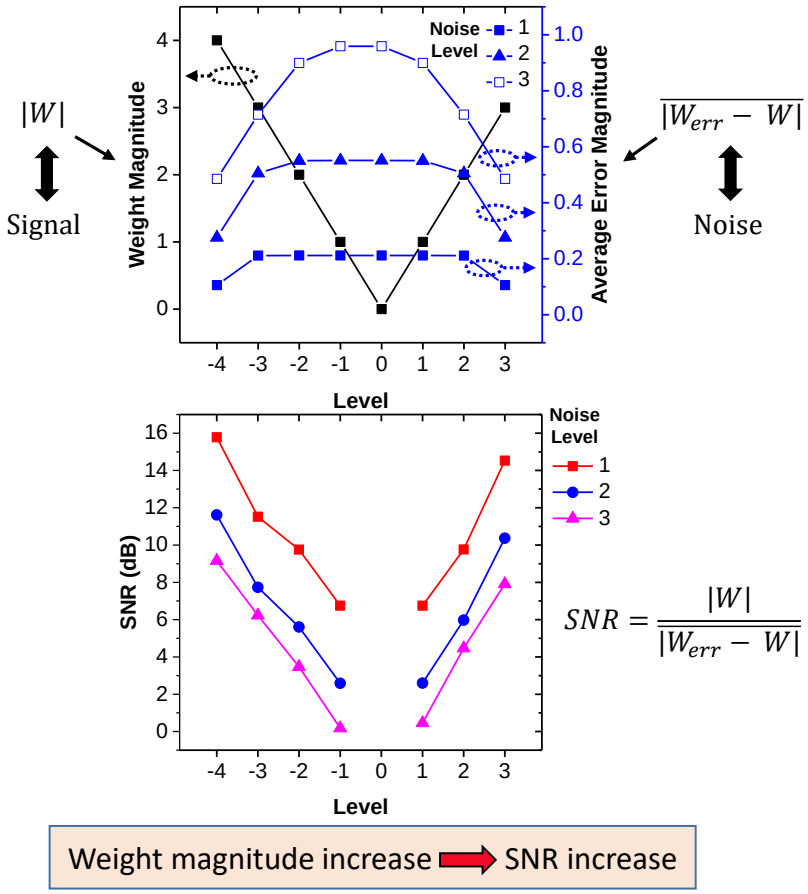


FIGURE 5.7: **Analysis of error training.** Weight Magnitude and Average Error Magnitude (Top) and Signal-to-Noise Ratio (Bottom) of the 8 NVM levels, for different noise levels (cf Fig. 5.1). As the level associated with the digital value "0" has a signal amplitude of "0", its SNR in dB is $-\infty$ and is not represented.

have the same magnitude. Hence, the outer levels have a higher SNR than the levels in the middle, meaning that the amplitude of the noise is relatively small compared to the amplitude of the signal they carry. Therefore, by increasing the magnitude of the weights, the SNR of synapses is increased, and hence the accuracy of the neural network is improved. In addition, the scaling decrease may be a consequence of the weight magnitude increase, allowing to keep the same magnitude of pre-activations.

These findings can allow further improvement of the performance of neural networks in the context of such hardware implementations. In these experiments, weights and scalings are found to be important for the network to increase its robustness to errors. However, these results depend on the quantization method, training procedure and error model. For instance, the quantization method, such as the choice of the digital values associated with the levels, has an impact on both the SNR of the levels and the learned weight distribution, and hence may be carefully considered. Moreover, having a NVM level associated with the value "0" may not be the optimal strategy under high noise level. Indeed, the value "0" is inherently very sensitive to noise due to its null magnitude. In addition, weight decay is a regularization method consisting in decaying the value of the weights at each training iteration that is frequently used to improve the accuracy of neural networks. Therefore, it could interfere with

TABLE 5.1: Strengths and weaknesses of different models (ANN vs SNN) and topologies (RNN vs CNN)

Model	Accuracy (no errors)	#Ops (consum.)	#Params (memory)	Robustness static errors	Robustness dyn. errors
CNN	++	–	+	+	+
SCNN	+	+	+	+	++
RNN	++	-	-	+	++
SRNN	+	++	-	+	++

"+"/"-" means better/worse.
Note that although the RNN and CNN topologies are chosen to have similar number of operations and parameters in these experiments, usually RNN implementations have more parameters than CNNs but lower number of operations.

the increase of weight magnitude associated with error-aware training. Finally, it also emphasizes the interest of including all possible parameters, such as scalings, in the optimization process, instead of setting them as hyperparameters, as some may have an unexpected impact on the robustness of neural networks to errors.

In addition, some aspects of the error-aware training were not investigated and could be of interest. For instance, the impact of pre-training the neural network in the error-agnostic condition before fine-tuning with the error-aware condition could be further studied. In addition, in the case of recurrent networks, the impact of the number of timesteps used in the forward, but also in the backward pass (if truncated BPTT is used) could have an impact on the robustness to errors.

5.2.4 Discussion

A general methodology to evaluate and improve the performance of neural networks in the context of synaptic weights implemented with multi-level NVMs was presented. Error-aware training is demonstrated to be very effective to enhance the robustness of neural networks to high error rates, making them suitable for multi-level NVM implementations. Moreover, two types of errors capturing the variability of NVMs, namely static and dynamic errors, have been distinguished and have shown a different impact on the accuracy of neural networks. In particular, SNNs and RNNs appear to be inherently more robust to dynamic than static errors, due to the nature of their computation using accumulation over time. In addition, they are found to be more robust to dynamic errors than CNNs. Hence, the SRNN SpikGRU is shown to be a promising solution for accurate and energy-efficient hardware implementations using noisy analog components, such as NVMs, as it achieves high robustness to errors while maintaining a high spike sparsity. Furthermore, for all neural networks, the weight distribution and scaling parameters learned were strongly impacted by the noise level during training.

In these experiments, the role of topology (CNN, RNN) and coding (SNN, ANN) was considered (see Table 5.1). Nevertheless, other factors may have an important impact on the robustness to errors. For instance, deeper networks are inherently less

robust to errors, as errors accumulate through layers, according to Yang et al., 2019a. Here, the effect of depth in time was considered (with RNNs). However, the case of deep networks in the spatial dimension (i.e. having more than a few layers) should be further investigated. Note that noise injection training strategies have shown high accuracy for CNNs with 20 layers (in Wan et al., 2022) or 34 layers (in Joshi et al., 2020), which suggests that this strategy is also effective for deeper networks in space. In addition, the role of the activation function could be studied. For instance, a higher inherent robustness to errors (without specific training) was observed for the RNN when using a *tanh* activation function rather than *ReLU* or *linear*, which suggests that bounded activation functions increase robustness to errors (in line with Malekzadeh et al., 2021). Moreover, the role of the binary activation function of SNNs in the robustness to errors was not investigated. Besides, robustness to errors may decrease with increasing task difficulty. In addition, only the weights of the neural networks have been simulated with the NVM model (i.e. with quantization and noise). Indeed, the weights often have the highest memory footprint compared to neuron-related data such as biases, membrane potentials, or time constants. Nevertheless, the implementation of the other data types with NVMs should be further investigated. This could allow to compare the impact of the noise on the different data types on the accuracy of the neural networks.

In this study, the NVM and error model is very general to be the most independent of the hardware implementation (such as choice of bit-cell implementation and NVM technology). However, this model can be adapted for each specific case. For instance, the impact of the combination of the two types of errors (static and dynamic) could be studied. Moreover, only errors related to the memories have been considered, while, depending on the implementation, other sources of errors can be added to the model (Moon et al., 2019; Vatajelu et al., 2019; Higuchi et al., 2022).

Finally, although specifically focused on the case of analog memories only used for weight storage, the methodology can be extended to the case of IMC. Note that the way of applying errors to the weights would be slightly different. Indeed, in this study, the errors are applied in a discretized way as levels are discretized (for instance, a weight can be read at level "0" or "1", but not at an intermediate value). On the contrary, in the case of IMC, the noisy analog values are directly used in the computation. Therefore, both the quantization method and error model should be modified. Indeed, the weights should be modeled with the analog values (instead of the digital values used here), for instance corresponding to the levels of conductance of the memory. Moreover, the noise should be modeled as a Gaussian noise directly applied on the analog values. In addition, sources of noise coming from the analog computation should be added to the model (Higuchi et al., 2022). Joshi et al., 2020 have shown that training neural networks with noise injection in the synaptic weights, even without a model for each specific source of errors, is still effective to achieve close to software accuracy on an IMC hardware implementation. Therefore, we believe that our main results still hold in this case, although this must be verified.

5.3 Case Study: Resistive Memories

Resistive memories (RRAMs) are promising to implement synaptic weights of neural networks (Yao et al., 2020). However, as other emerging NVMs, RRAMs are prone to

an important instrinsic operating variability, causing reading errors, impacting the accuracy of neural networks. At the system level, it is known that wide neural network topologies better tolerate variability due to their inherent redundancy. Therefore, overparametrized architectures are often used in this context (as in Hirtzlin et al., 2019). However, large topologies increase memory requirements. Therefore, high memory density becomes essential to limit the silicon footprint. In this context, replacing the 1 Transistor 1 Resistor (1T1R) memory architectures by denser 1 Selector 1 Resistor (1S1R) architectures, appears promising for increasing the density of the memory array. Indeed, NVM-based implementations of synaptic weights are often based on 1T1R architectures (Valentian et al., 2019; Joshi et al., 2020), where memory devices are accessed by individual selecting complementary metal oxide semi-conductor (CMOS) transistors. Conversely, in 1S1R architectures, the memory is co-integrated in series with a back-end selector. Hence, only one driver transistor per bit-line and word-line is needed, which allows to scale the bit-cell size from $40F^2$ to $4F^2$ (Minguet Lopez et al., 2021). In addition, high memory array capacity increases the latency. Therefore, high reading frequency is required to achieve fast neural network inference. However, Minguet Lopez et al., 2022 have shown that increasing the reading frequency induces a higher rate failure of Ovonic Threshold Switch (OTS) selectors (1S), hence degrading the BER.

In this section, the hardware model and training methodology developed in Section 5.2 is applied to a specific case of emerging NVM. RRAMs are considered with a 1S1R architecture. The 1S1R stack considered has binary capabilities (as opposed to multi-level approaches considered in Section 5.2). Hence, a BSNN using a simple FC topology is implemented. The robustness to errors of the BSNN is studied, and in particular the trade-off between the inference latency and accuracy is evaluated. Then figures of merit of the efficiency of the BSNN are presented, considering area and electrical consumption of the memory array.

5.3.1 Simulations of BSNNs with Resistive Memory Devices

Memory Devices

The 1S1R stack is composed of a HfO_2-based OxRAM memory device (1R) and an OTS selector (1S), and hence is called OxRAM+OTS. Hardware constraints (such as BER) are extracted from physical measurements performed on a memory array. The BSNN simulations were performed with varying BERs, corresponding to varying reading frequencies of the 1S1R. These experiments allow to evaluate the trade-off between the inference latency and accuracy. Then, two reading frequencies are particularly considered: 4MHz and 10MHz, associated with a BER of $\approx 5\text{x}10^{-2}$ and $1\text{x}10^{-1}$, respectively. Note that the OxRAM+OTS device is prone to both static and dynamic errors (as described in Section 5.2). Indeed, the variability of the RRAM is in part static (e.g. due to programming failures or conductance relaxation), and the errors associated with high frequency reading in the OTS are dynamic (they vary at each reading cycle).

Implementation of BSNNs

Considering a binary 1S1R (with two resistive stable states), the BSNN is implemented with binary weights (+1 or -1) and trained using the procedure described in Section 5.2.

A fully connected architecture with one hidden layer with varying size (X ∈ {512, 1024, 2048, 4096}) is considered to solve the digit classification task of the MNIST dataset (see Fig. 5.8). The MNIST images are converted to spikes before being processed by the BSNN.

Experiments are performed with a varying number of timesteps (from 1 to 10) to simulate the BSNN dynamics. If only one timestep is used, the images are converted to spikes using a threshold (equivalent to a binarization of the image). Only if the pixel intensity is above the threshold, an input spike is produced. Thus, the threshold allows to tune the sparsity of the input data. If multiple timesteps are used, the images are converted to spikes following a rate coding strategy. However, in these experiments, no important accuracy improvement was observed using more than one timestep. The more the timesteps, the more the input spikes, leading to a degraded sparsity of the BSNN. Therefore, in order to minimize the energy consumption of the BSNN implementation, only one timestep is used in these experiments. In this case, the BSNN is similar to a BNN (an ANN with binary activations). The difference is that the activations of the BSNN are +1 or 0 (while BNN usually have +1 or -1 activations). This allows to naturally benefit from the activation sparsity (null activations, meaning an absence of spikes) in an event-based implementation. Notably, the use of a unique SNN timestep means that there is no difference between static and dynamic errors. Therefore, no distinction is made in the following results.

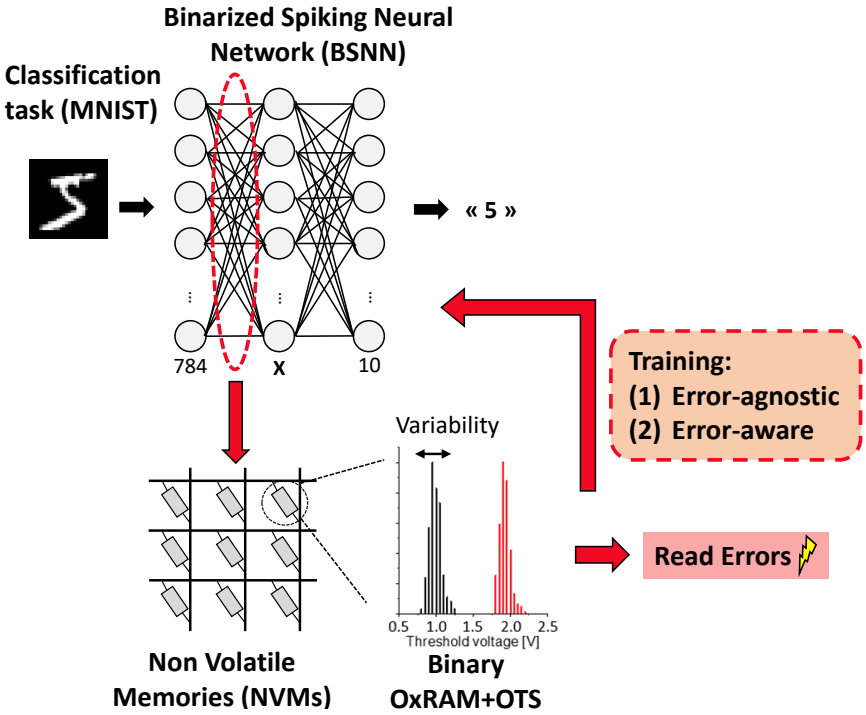


FIGURE 5.8: Implementation of the BSNN using OxRAM+OTS (1S1R) for synaptic weights storage. The BSNN is implemented with varying hidden layer size (X). For these experiments, only one timestep is used to simulate the SNN dynamics, therefore no distinction is made between static and dynamic errors in the fault model.

5.3.2 Improving the Robustness to Errors of BSNNs

The BSNN is trained in error-agnostic and error-aware conditions as described in Section 5.2. The accuracy of the BSNN with the two training conditions is shown in Fig. 5.9. In the case of error-agnostic training, although the BSNNs tolerate a certain BER (up to 1x10^{-2}), the accuracy is significantly degraded with a BER of 10^{-1} (corresponding to 10MHz reading frequency). Nevertheless, in the case of error-aware training, the accuracy at BER=10^{-1} is only slightly degraded (about 1 to 2%, depending on the topology). Thus, the effectiveness of error-aware training is again demonstrated. In addition, other factors impacting the robustness of the BSNN are evaluated: (1) the size of the hidden layer and (2) the choice of the BER used in error-aware training.

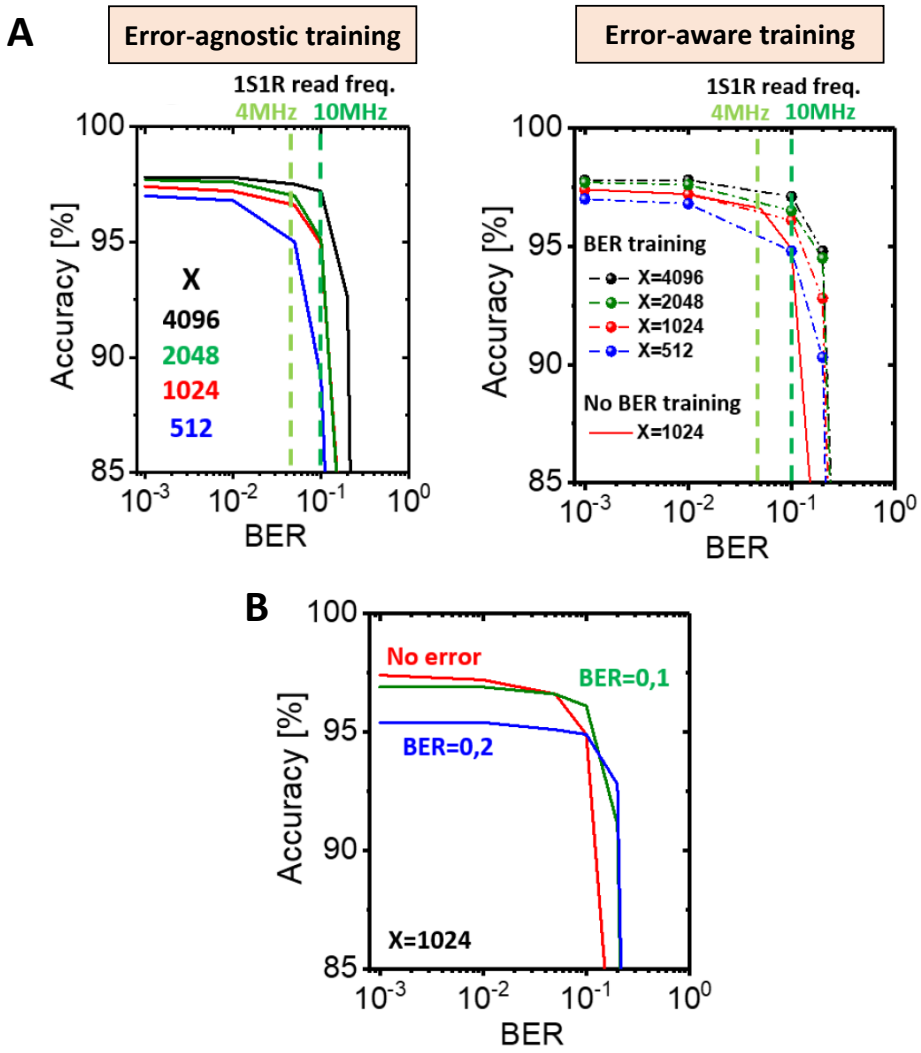


FIGURE 5.9: **Robustness to errors of the BSNN. A.** Accuracy of the BSNN with different hidden layer size (X) with the Bit Error Rate (BER) of the binary weights implemented using OxRAM+OTS in error-agnostic (left) and error-aware (right) training conditions. In the error-aware condition, the BER used for training is the same as the BER used for testing. The BER corresponding to the two considered reading frequencies are highlighted. **B.** Accuracy of the BSNN with the BER (during test) depending on the BER used for training in error-aware training condition (*No error* corresponds to error-agnostic training), for topology X=1024.

Impact of the Hidden Layer Size

The width of the neural network topology (hidden layer size in this case) is an important factor impacting the robustness to errors, both in error-agnostic and error-aware training conditions, as shown in Fig. 5.9. First, it is observed that, even in error-free testing, the wider the BSNN, the higher its accuracy. In these experiments, the maximal accuracy (without errors) reached by the BSNNs is from 97% to 98%, from the smallest to the biggest topology. Second, the wider the neural network, the higher the robustness to errors, both in error-agnostic and error-aware training. For instance, in error-agnostic training, while the accuracy of the largest BSNN ($X = 4096$) is only degraded by 1% in the case of 10MHz reading frequency (BER=1x10^{-1}), the accuracy of the smallest BSNN ($X = 512$) is highly degraded (by 8%). In addition, the same behavior is observed in error-aware training. Indeed, although the accuracy of the smallest BSNN is less degraded than in the error-agnostic case (2% loss), it is still more degraded than that of the largest BSNN.

Impact of the BER in Error-Aware Training

In Section 5.2, an hypothesis was made on the best noise level to use in error-aware training to obtain the highest accuracy during test with errors. In particular, it is assumed that the BER used in the error-aware training must be equal to the BER used in the testing phase. In these experiments, the choice of the BER used during training is considered (see Fig. 5.9). It is observed that the optimal choice of BER during training is the one approximately corresponding to the BER used for testing, confirming the previous hypothesis in this context. Moreover, it is important to note that a BSNN trained with a high BER achieves a lower accuracy in error-free and low-BER tests, than a BSNN trained with no errors or a lower BER. For instance, the BSNN trained at BER=0.2 achieves higher accuracy at BER=0.2 than the BSNN trained at BER=0.1, but lower accuracy at BER=0.1. This shows that the optimization process favored robustness over high accuracy. Therefore, it is important to correctly estimate the BER of the devices, in order to adjust the training process to reach maximal performance.

5.3.3 Improving the Efficiency of BSNNs

Furthermore, the BSNN is optimized to improve the efficiency of the hardware implementation. In particular, area and electrical consumption of the memory array are considered (see Fig. 5.10). In this section, not only the specific noise of the devices is considered, but also the noise induced by non-idealities at the array level. Indeed, large memory arrays are prone to the IR voltage drop phenomenon (voltage drop due to current flowing through a resistor), which also participate in degrading the BER (see Minguet Lopez et al., 2022 for further details).

Trade-off between Area and Accuracy

The higher the number of parameters of the neural network, the larger the memory array, considering that a single memory array is used to store all the weights. Therefore, large neural networks not only require higher area, but are also more prone to the IR

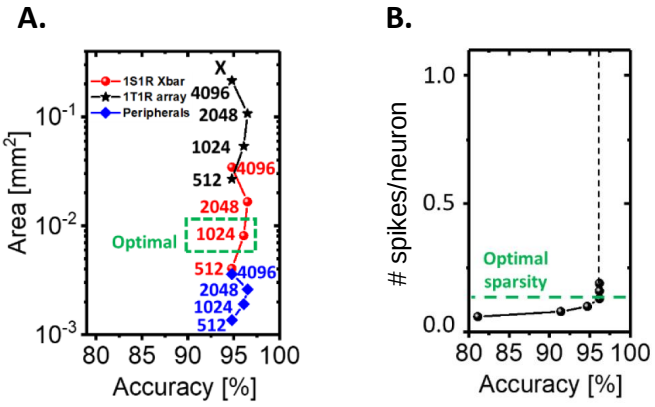


FIGURE 5.10: **Optimization of the efficiency of the BSNN. A.** Trade-off between the area of the memory array and accuracy of BSNNs with different hidden layer size. The area of the 1S1R array, the equivalent 1T1R array, and the peripherals of the 1S1R array are shown (details of the computation can be found in Minguet Lopez et al., 2022). **B.** BSNN activity (average number of spikes per neuron per inference) depending on the BSNN accuracy (topology X=1024).

drop phenomenon. In these experiments, the size of the memory array is determined by the hidden layer size (X) of the neural network.

The trade-off between area and accuracy (taking into account the IR drop phenomenon) for the different topologies is shown in Fig. 5.10. Considering the IR drop phenomenon, at a target reading frequency of 4MHz, the resulting BER is not anymore $5x10^{-2}$ for the topology X=4096 (contrary to what is assumed in Section 5.3.2). Therefore, the resulting accuracy of the BSNN with X=4096 becomes lower than that of smaller BSNNs. In this context, the topology X=1024 offers a good compromise between accuracy and memory area. Moreover, the area of the 1S1R memory array is compared to that of an implementation using a 1T1R architecture. Compared to the 1T1R, the 1S1R implementation shows an order of magnitude improvement in area. The area of the peripherals of the 1S1R array (peripheral circuitry needed to read and program the memory array) is also shown for comparison.

Trade-off between Energy Efficiency and Accuracy

The electrical consumption of the memory array depends on the number of weight read per inference. Therefore, in the case of a BSNN implementation, the BSNN sparsity (measured with the spiking activity, i.e. average number of spikes per inference) directly impacts the electrical consumption of the memory array. In these experiments, the input-to-hidden layer is almost two orders of magnitude bigger than the hidden-to-output layer. Therefore, the network sparsity is mostly determined by the input spikes received by the input-to-hidden layer, i.e. the sparsity of the input data (MNIST images converted to spikes). The input sparsity can be tuned by adjusting the threshold used in the pixel-to-spike conversion process (as explained in Section 5.3.1).

The accuracy of the BSNN depending on the sparsity is shown in Fig. 5.10, in the case of the topology X=1024. In these experiments, the highest sparsity leading to the maximal accuracy is 0.13 spikes per neuron per inference. Therefore, compared to a BNN (where there is no activation sparsity) with equivalent accuracy on the task of

interest (as in Minguet Lopez et al., 2021), the electrical consumption is decreased by almost one order of magnitude.

5.3.4 Discussion

Resistive memories with 1S1R architectures, such as OxRAM+OTS, can allow dense implementations of SNNs. Using the MNIST task, the BSNN demontrated a high tolerance to errors, further validating the error-aware training strategy proposed in Section 5.2. In particular, an appropriate choice of noise level during training and a wide neural network topology are demonstrated to be important factors for achieving a high robustness to noise. In addition, the efficiency of the BSNN implementation is optimized. First, a moderate topology width is shown to achieve a good trade-off between accuracy and area (and avoid noise induced by large memory array). Second, the sparsity of the BSNN is chosen to minimize the energy consumption while preserving a high accuracy, leveraging an event-based implementation.

Notably, these results show two benefits of wide neural network topologies. First, wider topologies allow to reach a higher maximal accuracy in the case of highly-quantized weights (such as binary weights). Indeed, the reduced precision of the weights can be partly compensated by having more neurons, up to some extent (too many neurons result in a high number of parameters which can make the neural network overfit). This is in line with experiments on SNNs with full-precision weights, where wide topologies are used to compensate for the reduced precision of activations, as discussed in Section 2.3. Second, wider topologies also have a higher robustness to errors in weights, in both error-agnostic and error-aware training conditions. Indeed, wider topologies have more redundancy, which decreases the impact of errors (errors can compensate each other more easily). Nevertheless, if a single array is used to implement a layer, the width of the layer will be limited by the array capacity (considering the IR drop phenomenon).

5.4 Conclusion

A hardware fault model for simulating implementations with single- and multi-level NVMs was proposed. The model is meant to be technology-independent and, in principle, applicable to all kind of emerging analog NVMs. Then, a training methodology adapted to highly-quantized and faulty weights was presented. The robustness of different topologies (CNN and RNN) and coding strategies (ANN and SNN) was compared in the case of multi-level NVMs. The error-aware training strategy is shown to effectively allow neural networks to be robust to high error rates, with only a small decrease in accuracy compared to the error-free case. In particular, the SpikGRU model is shown to be promising for such implementations, as it benefits from a particular robustness to dynamic errors, as well as a higher energy efficiency than the other models due to the high spike sparsity, in line with the results from Chapter 4.

Then, a case study using resistive memories with 1S1R architectures with single-level capabilities was investigated. Physical measurements performed on a memory array allowed to model realistic hardware constraints. The general fault model and training methodology developed were applied to train a FC Binary SNN. The results

validate the proposed approach and emphasize the role of the choice of noise level during training and network width for the robustness. Issues related to implementations using analog memory arrays were highlighted, such as the trade-off between accuracy and area due to the IR drop phenomenon, and, in the case of 1S1R architectures using OTS selectors, the trade-off between accuracy and latency due to the erratic switching behavior of the OTS.

Future work will consider more challenging tasks to validate the approach, using experimental hardware characteristics in the case of multi-level NVMs and more sophisticated neural network architectures with CNN or RNN topologies (such as SpikGRU). In particular, the conclusions on robustness, sparsity and topology should be further investigated in these more challenging contexts.

In addition, the benefits from using an analog implementation compared to a fully-digital one should be further studied. However, it is beyond the scope of this work to estimate the benefit in terms of energy consumption, as it depends on the circuit implementation (which depends in particular on whether an IMC implementation is considered or not), while only the memory aspect has been considered here. In addition, the overall energy footprint should be considered, including the static energy consumption, and, more importantly, the accesses to the external non-volatile storage memory (in the case of the fully-digital implementation, as the on-chip memory is volatile). Conversely, in this thesis, we have limited the study to the dynamic energy consumption and the on-chip elements. in the interests of comparing ANN and SNN processing modes. Notably, a major advantage of the emerging NVMs considered in this chapter (such as RRAMs) over volatile memories (such as DRAMs and SRAMs) is to remove this dependency on external storage, an advantage that is not only measurable in terms of energy consumption.

Chapter 6

Summary and Perspectives

With the widespread use of AI in every sectors of the economy, it is urgent to find efficient systems capable of performing AI inference and training at low cost and with a low environmental footprint. To achieve this goal in spite of the increasing complexity of designing such systems, the co-development of hardware and software is necessary. SNN algorithms, based on the asynchronous accumulation of sparse spike-based events, can be leveraged in event-based hardware implementations to minimize the energy consumption and latency per inference. Moreover, high accuracy can be achieved by leveraging spatio-temporal accumulations in high precision neuronal states, despite the highly-quantized activations. Therefore, SNNs implemented on event-based neuromorphic hardware could lead to accurate and efficient intelligent systems.

Nevertheless, the energy efficiency of SNNs has often been overestimated. This is partly due to a lack of consideration for the underlying hardware implementation. In particular, many works have considered the replacement of MAC operations with multiple AC operations as an energy benefit, while neglecting the significant impact of memory accesses. Moreover, fully-analog SNN implementations with infinite temporal precision have also been used as an argument towards SNN efficiency. However, the low maturity of such implementations, as well as the lack of efficient temporal coding learning algorithms, makes such efficient implementations difficult to achieve so far.

In this thesis, we have followed a hardware-aware approach to drive the developments of SNN algorithms. This has led to models and algorithms with the objective of improving both accuracy and efficiency of SNNs for their implementation on dedicated neuromorphic accelerators, possibly digital or analog.

We have first analyzed existing SNN algorithms, in particular considering training strategies aiming at improving their accuracy and the efficiency of their implementation. Notably, we have observed the important impact of network architecture width and encoding layer on the accuracy-latency trade-off in SNNs. However, existing methods for comparing the energy efficiency of ANNs and SNNs are limited.

Consequently, we have proposed a high-fidelity model of the energy efficiency of SNNs and ANNs on neural network accelerators. We have derived theoretical lower and upper bounds for their relative energy efficiency, as well as realistic models based on the Eyeriss accelerator. This study demonstrates the primary impact of spike sparsity on the efficiency of event-based SNN implementations. In addition, we show that high precision neuronal variables and associated MAC operations have a relatively low impact on the global SNN footprint compared to synaptic operations. Moreover, the ability to leverage high data reuse is one of the main advantage of ANNs that cannot be leveraged in event-based implementations. Therefore, SNNs could bring higher

benefits for network topologies offering fewer opportunities of data reuse, such as fully connected and recurrent topologies.

These results have led to the proposition of the SpikGRU model, leveraging the accuracy of gated recurrent units using high precision neuronal variables, and the efficiency of sparse spiking activations. The use of recurrent topologies, with no additional timestep for the SNN, for processing spatio-temporal data has allowed to achieve a high spike sparsity. In addition, sparsity was effectively enhanced through gradient descent. The extension of the model of the dynamic energy consumption to the case of gated recurrent topologies validates the proposed approach. Indeed, it shows that SpikGRU can be more energy efficient than its ANN equivalent, even when considering optimal ANN implementations.

In addition, analog implementations could lead to further benefits. In particular, the integration of emerging NVMs on chip could allow to remove the dependency on external data storage, and hence reducing the global system energy footprint. Nevertheless, analog components are prone to variability, which imposes constraints on the robustness of neural networks. Therefore, we have proposed a hardware fault model to simulate the non-idealities of emerging NVMs considering single- and multi-level implementations. Then, a training methodology adapted to highly-quantized and noisy weights was presented. The performance of ANNs and SNNs with different topologies (CNN and RNN) under such constraints was demonstrated to be satisfactory. In particular, the SpikGRU model was shown to be very robust to noise, despite its high activation sparsity, and hence makes it promising to use in such context. In addition, the approach was evaluated in the case of resistive memories with 1S1R architectures, which promise higher memory density than traditional architectures, and therefore are of particular interest for wide network topologies.

Based on these considerations, we make some suggestions in the perspective of future hardware-algorithm co-developments. First, a better compromise could be found between the accuracy provided by high precision data and operations, and the efficiency of event-based implementations. For instance, hybrid ANN-SNN implementations (with some ANN layers and some SNN layers) could provide additional benefits, building on existing work in mixed precision ANN training. Alternatively, the hybridization between SNN and ANN neurons could allow to leverage event-based processing while relaxing the high quantization imposed on spiking activations. In addition, throughout this thesis, we have seen the impact of using wide network topologies on many of the parameters studied, such as spike sparsity, tolerance to weight quantization and robustness to errors. However, this comes at the cost of higher memory requirements, higher static consumption, and this technique may be not scalable for already large network topologies. In this context, synaptic weight pruning in SNNs, in particular in wide recurrent topologies, could potentially solve this problem.

Considering the future of analog implementations, multi-level NVMs promise to achieve high memory density by using only one device per weight encoded. However, multi-level programming often comes with a higher noise level and hence, the optimal number of levels maximizing the accuracy should be investigated. Besides, exploiting temporal coding in analog hardware could potentially bring the efficiency of SNNs to another level. However, many challenges must first be overcome, both at the software and hardware level.

Finally, this thesis has tackled the challenge of efficient inference of neural networks on dedicated accelerators for edge applications. Nevertheless, the problem of on-chip training is becoming increasingly important, not only to improve the training efficiency, but also to enable AI systems to continue learning once deployed at the edge. In addition, we have seen the limitations of off-chip SNN training with BPTT for achieving high accuracy and high convergence speed. Indeed, as the networks go deeper and task difficulty increases, the high spike sparsity, on the one hand, and the high number of timesteps, on the other hand, make the training with BPTT challenging. Therefore, lighter alternatives to BPTT could not only improve SNN training, but also enable more efficient on-chip training. On-chip training will also bring benefits for analog implementations. Indeed, the specific static and dynamic noise of analog components could be naturally accounted for during the training phase.

Hence, as we have shown the effectiveness of hardware-algorithm co-development in achieving efficient solutions for the problem of inference, we argue that this approach should be pursued to address the challenges of on-chip training. In doing so, we hope to pave the way for efficient AI systems at the edge.

Bibliography

Amir, Arnon, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha (July 2017). "A Low Power, Fully Event-Based Gesture Recognition System". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI: IEEE, pp. 7388–7397. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.781.

Amrouch, Hussam, Nan Du, Anteneh Gebregiorgis, Said Hamdioui, and Ilia Polian (2021). "Towards Reliable In-Memory Computing:From Emerging Devices to Post-von-Neumann Architectures". In: *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6. DOI: 10.1109/VLSI-SoC53125.2021.9606966.

Anumula, Jithendar, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu (2018a). "Feature Representations for Neuromorphic Audio Spike Streams". In: *Frontiers in Neuroscience* 12. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00023. (Visited on 10/14/2020).

— (2018b). "Feature Representations for Neuromorphic Audio Spike Streams". In: *Frontiers in Neuroscience* 12. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00023. (Visited on 10/14/2020).

Auge, Daniel, Julian Hille, Etienne Mueller, and Alois Knoll (2021). "A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks". In: *Neural Process. Lett.* 53.6, 4693–4710. ISSN: 1370-4621. DOI: 10.1007/s11063-021-10562-2.

Baktha, Kiran and B. K. Tripathy (2017). "Investigation of recurrent neural networks in the field of sentiment analysis". In: *2017 International Conference on Communication and Signal Processing (ICCSP)*, pp. 2047–2050. DOI: 10.1109/ICCSP.2017.8286763.

Balatti, S., S. Larentis, D. C. Gilmer, and D. Ielmini (2013). "Multiple Memory States in Resistive Switching Devices Through Controlled Size and Orientation of the Conductive Filament". In: *Advanced Materials* 25.10, pp. 1474–1478. DOI: https://doi.org/10.1002/adma.201204097.

Bellec, Guillaume, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass (Dec. 2018a). "Long short-term memory and learning-to-learn in networks of spiking neurons". In: *arXiv:1803.09574 [cs, q-bio]*.

Bellec, Guillaume, Darjan Salaj, Anand Subramoney, Robert A. Legenstein, and Wolfgang Maass (2018b). "Long short-term memory and Learning-to-learn in networks of spiking neurons". In: *Advances in Neural Information Processing Systems: NeurIPS*, pp. 795–805.

Bellec, Guillaume, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass (July 2020). "A solution to the learning dilemma for recurrent networks of spiking neurons". en. In: *Nature Communications* 11.1.

Number: 1 Publisher: Nature Publishing Group, p. 3625. ISSN: 2041-1723. DOI: 10.1038/s41467-020-17236-y. (Visited on 09/21/2020).

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. DOI: 10.1109/72.279181.

Bengio, Yoshua, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin (Aug. 8, 2016). "Towards Biologically Plausible Deep Learning". In: *arXiv:1502.04156 [cs]*. arXiv: 1502.04156. URL: http://arxiv.org/abs/1502.04156 (visited on 09/11/2020).

Bengio, Yoshua, Nicholas Leonard, and Aaron Courville (2013). "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *arXiv:1308.3432 [cs.LG]*.

Bhattacharjee, Abhiroop, Youngeun Kim, Abhishek Moitra, and Priyadarshini Panda (2022). "Examining the Robustness of Spiking Neural Networks on Non-Ideal Memristive Crossbars". In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. ISLPED '22. Boston, MA, USA. ISBN: 9781450393546. DOI: 10.1145/3531437.3539729.

Blouw, Peter and Chris Eliasmith (2020a). "Deep convolutional spiking neural networks for keyword spotting". In: *Proceedings of Interspeech*, 2557–2561.

— (2020b). "Event-Driven Signal Processing with Neuromorphic Computing Systems". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8534–8538. DOI: 10.1109/ICASSP40776.2020.9053043.

Boahen, K.A. (May 2000). "Point-to-point connectivity between neuromorphic chips using address events". en. In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47.5, pp. 416–434. ISSN: 10577130. DOI: 10.1109/82.842110. (Visited on 11/09/2021).

Bohnstingl, Thomas, Anja Šurina, Maxime Fabre, Yiğit Demirağ, Charlotte Frenkel, Melika Payvand, Giacomo Indiveri, and Angeliki Pantazi (2022). "Biologically-inspired training of spiking recurrent neural networks with neuromorphic hardware". In: *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 218–221. DOI: 10.1109/AICAS54282.2022.9869963.

Bohte, Sander M., Joost N. Kok, and Han La Poutré (2002). "Error-backpropagation in temporally encoded networks of spiking neurons". In: *Neurocomputing* 48.1, pp. 17–37. ISSN: 0925-2312. DOI: 10.1016/S0925-2312(01)00658-0.

Bose, Sumon Kumar, Jyotibdha Acharya, and Arindam Basu (2019). "Is my Neural Network Neuromorphic? Taxonomy, Recent Trends and Future Directions in Neuromorphic Engineering". In: *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 1522–1527. DOI: 10.1109/IEEECONF44664.2019.9048891.

Bu, Tong, Jianhao Ding, Zhaofei yu, and Tiejun Huang (June 2022a). "Optimized Potential Initialization for Low-Latency Spiking Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36, pp. 11–20. DOI: 10.1609/aaai.v36i1.19874.

Bu, Tong, Wei Fang, Jianhao Ding, Penglin Dai, Zhaofei Yu, and Tiejun Huang (2022b). "Optimal ANN-SNN Conversion for High-accuracy and Ultra-low-latency Spiking Neural Networks". In: *International Conference on Learning Representations*.

Burel, Stéphane, Adrian Evans, and Lorena Anghel (2022). "MOZART+: Masking Outputs With Zeros for Improved Architectural Robustness and Testing of DNN Accelerators". In: *IEEE Transactions on Device and Materials Reliability* 22.2, pp. 120–128. DOI: 10.1109/TDMR.2022.3159089.

Cao, Yongqiang, Yang Chen, and Deepak Khosla (May 2015). "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". en. In: *International Journal of Computer Vision* 113.1, pp. 54–66. ISSN: 1573-1405. DOI: 10.1007/s11263-014-0788-3. (Visited on 04/21/2022).

Castagnetti, Andrea, Alain Pegatoquet, and Benoît Miramond (2023). "Trainable quantization for Speedy Spiking Neural Networks". In: *Frontiers in Neuroscience* 17. ISSN: 1662-453X. DOI: 10.3389/fnins.2023.1154241.

Chakraborty, Biswadeep and Saibal Mukhopadhyay (2023). "Heterogeneous recurrent spiking neural network for spatio-temporal classification". In: *Frontiers in Neuroscience* 17. ISSN: 1662-453X. DOI: 10.3389/fnins.2023.994517. URL: https://www.frontiersin.org/articles/10.3389/fnins.2023.994517.

Chan, Vincent, Shih-Chii Liu, and Andr van Schaik (Jan. 2007). "AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.1. IEEE Transactions on Circuits and Systems I: Regular Papers, pp. 48–59. ISSN: 1558-0806. DOI: 10.1109/TCSI.2006.887979.

Chen, Yingyi, Qianqian Cheng, Yanjun Cheng, Hao Yang, and Huihui Yu (2018). "Applications of Recurrent Neural Networks in Environmental Factor Forecasting: A Review". In: *Neural Computation* 30.11, pp. 2855–2881. DOI: 10.1162/neco_a_01134.

Chen, Yu-Hsin, Joel Emer, and Vivienne Sze (June 2016). "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks". en. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. Seoul, South Korea: IEEE, pp. 367–379. ISBN: 978-1-4673-8947-1. DOI: 10.1109/ISCA.2016.40. (Visited on 11/09/2021).

Chen, Yu-Hsin, Tushar Krishna, Joel S. Emer, and Vivienne Sze (Jan. 2017). "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". en. In: *IEEE Journal of Solid-State Circuits* 52.1, pp. 127–138. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/JSSC.2016.2616357. (Visited on 11/09/2021).

Chen, Yu-Hsin, Tien-Ju Yang, Joel S. Emer, and Vivienne Sze (June 2019). "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices". en. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.2, pp. 292–308. ISSN: 2156-3357, 2156-3365. DOI: 10.1109/JETCAS.2019.2910232. (Visited on 11/09/2021).

Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (Oct. 2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

Comşa, Iulia-Maria, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala (2022). "Temporal Coding in Spiking Neural Networks With Alpha Synaptic Function: Learning With Backpropagation". In: *IEEE*

Transactions on Neural Networks and Learning Systems 33.10, pp. 5939–5952. DOI: 10.1109/TNNLS.2021.3071976.

Corradi, Federico, Guido Adriaans, and Sander Stuijk (Jan. 2021). "Gyro: A Digital Spiking Neural Network Architecture for Multi-Sensory Data Analytics". en. In: *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*. Budapest Hungary: ACM, pp. 9–15. ISBN: 978-1-4503-8952-5. DOI: 10.1145/3444950.3444951. (Visited on 11/09/2021).

Courbariaux, Matthieu, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio (2016). *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*. DOI: 10.48550/ARXIV.1602.02830.

Cramer, Benjamin, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke (2020). "The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.3044364.

Dampfhoffer, Manon, Thomas Mesquida, Emmanuel Hardy, Alexandre Valentian, and Lorena Anghel (2023a). "Leveraging Sparsity with Spiking Recurrent Neural Networks for Energy-Efficient Keyword Spotting". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10097174.

Dampfhoffer, Manon, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel (2022). "Investigating Current-Based and Gating Approaches for Accurate and Energy-Efficient Spiking Recurrent Neural Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2022*. Ed. by Elias Pimenidis, Plamen Angelov, Chrisina Jayne, Antonios Papaleonidas, and Mehmet Aydin. Springer Nature Switzerland, pp. 359–370. ISBN: 978-3-031-15934-3.

— (2023b). "Are SNNs Really More Energy-Efficient Than ANNs? an In-Depth Hardware-Aware Study". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 7.3, pp. 731–741. DOI: 10.1109/TETCI.2022.3214509.

— (2023c). "Backpropagation-Based Learning Techniques for Deep Spiking Neural Networks: A Survey". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–16. DOI: 10.1109/TNNLS.2023.3263008.

Dampfhoffer, Manon, Joel Minguet Lopez, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel (2023d). "Improving the Robustness of Neural Networks to Noisy Multi-Level Non-Volatile Memory-based Synapses". In: *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN54540.2023.10191804.

Dan, Yang and Mu-Ming Poo (2006). "Spike Timing-Dependent Plasticity: From Synapse to Perception". In: *Physiological Reviews* 86.3, pp. 1033–1048. ISSN: 0031-9333, 1522-1210. DOI: 10.1152/physrev.00030.2005. (Visited on 07/22/2021).

Davidson, Simon and Steve B. Furber (2021). "Comparison of Artificial and Spiking Neural Networks on Digital Hardware". English. In: *Frontiers in Neuroscience* 15. ISSN: 1662-453X. DOI: 10.3389/fnins.2021.651141. (Visited on 04/28/2021).

Davies, Mike, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang (2018). "Loihi: A Neuromorphic

Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1, pp. 82–99. DOI: 10.1109/MM.2018.112130359.

Davies, Mike, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A. Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R. Risbud (2021). "Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook". In: *Proceedings of the IEEE* 109.5, pp. 911–934. DOI: 10.1109/JPROC.2021.3067593.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

Deng, Lei, Yujie Wu, Xing Hu, Ling Liang, Yufei Ding, Guoqi Li, Guangshe Zhao, Peng Li, and Yuan Xie (Jan. 2020). "Rethinking the performance comparison between SNNs and ANNs". In: *Neural Networks* 121, pp. 294–307. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.09.005. (Visited on 09/17/2020).

Deng, Shikuang, Yuhang Li, Shanghang Zhang, and Shi Gu (2022). "Temporal Efficient Training of Spiking Neural Network via Gradient Re-weighting". In: *International Conference on Learning Representations*.

Diehl, Peter U., Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer (July 2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015 International Joint Conference on Neural Networks (IJCNN). Killarney, Ireland: IEEE, pp. 1–8. ISBN: 978-1-4799-1960-4. DOI: 10.1109/IJCNN.2015.7280696. (Visited on 07/22/2021).

Doevenspeck, J., K. Garello, S. Rao, F. Yasin, S. Couet, G. Jayakumar, A. Mallik, S. Cosemans, P. Debacker, D. Verkest, R. Lauwereins, W. Dehaene, and G.S. Kar (2021). "Multi-pillar SOT-MRAM for Accurate Analog in-Memory DNN Inference". In: *2021 Symposium on VLSI Technology*, pp. 1–2.

Duan, Chaoteng, Jianhao Ding, Shiyan Chen, Zhaofei Yu, and Tiejun Huang (2022). "Temporal Effective Batch Normalization in Spiking Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 34377–34390. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/de2ad3ed44ee4e675b3be42aa0b615d0-Paper-Conference.pdf.

Eshraghian, Jason K., Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu (Jan. 2022). "Training Spiking Neural Networks Using Lessons From Deep Learning". In: *arXiv:2109.12894 [cs]*.

Esteva, Andre, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun (Feb. 1, 2017). "Dermatologist-level classification of skin cancer with deep neural networks". In: *Nature* 542.7639, pp. 115–118. ISSN: 1476-4687. DOI: 10.1038/nature21056.

Fang, Haowen, Amar Shrestha, Ziyi Zhao, and Qinru Qiu (July 2020a). "Exploiting Neuron and Synapse Filter Dynamics in Spatial Temporal Learning of Deep Spiking Neural Network". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence {IJCAI-PRICAI-20}. Yokohama, Japan, pp. 2799–2806. ISBN: 978-0-9992411-6-5. DOI: 10.24963/ijcai.2020/388. (Visited on 09/14/2021).

Fang, Wei, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Timothée Masquelier, Yonghong Tian, and other contributors (2020b). *SpikingJelly*. https://github.com/fangwei123456/spikingjelly.

Fang, Wei, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian (2021a). "Deep Residual Learning in Spiking Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 21056–21069.

Fang, Wei, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian (2021b). "Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2641–2651. DOI: 10.1109/ICCV48922.2021.00266.

Fu, Qiang and Hongbin Dong (2021). "An ensemble unsupervised spiking neural network for objective recognition". In: *Neurocomputing* 419, pp. 47–58. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2020.07.109.

Garg, Isha, Sayeed Shafayet Chowdhury, and Kaushik Roy (Oct. 2020). "DCT-SNN: Using DCT to Distribute Spatial Information over Time for Learning Low-Latency Spiking Neural Networks". In: *arXiv:2010.01795 [cs, stat]*. URL: http://arxiv.org/abs/2010.01795 (visited on 04/30/2021).

Garofolo, J. S., L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren (1993). *DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM*.

Gerstner, W., W. M. Kistler, R. Naud, and L. Paninski (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press.

Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik (2014). "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

Grossi, A., E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. El Hajjam, R. Crochemore, J.F. Nodin, P. Olivo, and L. Perniola (2016). "Fundamental variability limits of filament-based RRAM". In: *2016 IEEE International Electron Devices Meeting (IEDM)*, pp. 4.7.1–4.7.4. DOI: 10.1109/IEDM.2016.7838348.

Guo, Wenzhe, Mohammed E. Fouda, Ahmed M. Eltawil, and Khaled Nabil Salama (2022). "Efficient Neuromorphic Hardware Through Spiking Temporal Online Local Learning". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.11, pp. 1642–1653. DOI: 10.1109/TVLSI.2022.3208191.

Göltz, J., A. Baumbach, S. Billaudelle, A. F. Kungl, O. Breitwieser, K. Meier, J. Schemmel, L. Kriener, and M. A. Petrovici (Mar. 2020). "Fast and deep neuromorphic learning with first-spike coding". In: *Proceedings of the Neuro-inspired Computational Elements Workshop*. NICE '20. New York, NY, USA, pp. 1–3. ISBN: 978-1-4503-7718-8. DOI: 10.1145/3381755.3381770. (Visited on 10/26/2020).

Han, Bing and Kaushik Roy (2020a). "Deep Spiking Neural Network: Energy Efficiency Through Time Based Coding". In: *Computer Vision – ECCV 2020*. Cham: Springer International Publishing, pp. 388–404. ISBN: 978-3-030-58607-2.

Han, Bing, Gopalakrishnan Srinivasan, and Kaushik Roy (2020b). "RMP-SNN: Residual Membrane Potential Neuron for Enabling Deeper High-Accuracy and Low-Latency Spiking Neural Network". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13555–13564. DOI: 10.1109/CVPR42600.2020.01357.

He, Kaiming, X. Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Higuchi, Kazuhide, Chihiro Matsui, and Ken Takeuchi (2022). "Investigation of Memory Non-Ideality Impacts on Non-Volatile Memory Based Computation-in-Memory AI Inference by Comprehensive Simulation Platform". In: *2022 IEEE Silicon Nanoelectronics Workshop (SNW)*, pp. 1–2. DOI: 10.1109/SNW56633.2022.9889067.

Hinton, Geoffrey (2012). "Neural networks for machine learning, coursera". In: *Coursera, video lectures*.

Hinton, Geoffrey, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury (2012). "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6, pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

Hirtzlin, T., M. Bocquet, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz (2019). "Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks". In: *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 288–292. DOI: 10.1109/AICAS.2019.8771544.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Hodgkin, A. L. and A. F. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117.4, pp. 500–544. ISSN: 1469-7793. DOI: https://doi.org/10.1113/jphysiol.1952.sp004764.

Horowitz, Mark (2014). "Computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323.

Hung, Je-Min, Yen-Hsiang Huang, Sheng-Po Huang, Fu-Chun Chang, Tai-Hao Wen, Chin-I Su, Win-San Khwa, Chung-Chuan Lo, Ren-Shuo Liu, Chih-Cheng Hsieh, Kea-Tiong Tang, Yu-Der Chih, Tsung-Yung Jonathan Chang, and Meng-Fan Chang (2022). "An 8-Mb DC-Current-Free Binary-to-8b Precision ReRAM Nonvolatile Computing-in-Memory Macro using Time-Space-Readout with 1286.4-21.6TOPS/W for Edge-AI Devices". In: *2022 IEEE International Solid- State Circuits Conference (ISSCC)*. Vol. 65, pp. 1–3. DOI: 10.1109/ISSCC42614.2022.9731715.

Ielmini, Daniele and Stefano Ambrogio (Dec. 2019). "Emerging neuromorphic devices". In: *Nanotechnology* 31.9, p. 092001. DOI: 10.1088/1361-6528/ab554b.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15, 448–456.

Izhikevich, E.M. (2003). "Simple model of spiking neurons". In: *IEEE Transactions on Neural Networks* 14.6, pp. 1569–1572. ISSN: 1941-0093. DOI: 10.1109/TNN.2003.820440.

Jin, Yingyezhe, Wenrui Zhang, and Peng Li (Dec. 2018). "Hybrid macro/micro level backpropagation for training deep spiking neural networks". In: *Proceedings of the*

32nd International Conference on Neural Information Processing Systems. NIPS'18. Red Hook, NY, USA, pp. 7005–7015. (Visited on 09/14/2021).

Joshi, Vinay, Manuel Le Gallo, Simon Haefeli, Irem Boybat, S. R. Nandakumar, Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou (Dec. 2020). "Accurate deep neural network inference using computational phase-change memory". English. In: *Nat Commun* 11.1. ISSN: 2041-1723. DOI: 10.1038/s41467-020-16108-9.

Joubert, A., B. Belhadj, O. Temam, and R. Héliot (2012). "Hardware spiking neurons design: Analog or digital?" In: *Proceedings IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–5. DOI: 10.1109/IJCNN.2012.6252600.

Jung, Seungchul, Hyungwoo Lee, Sungmeen Myung, Hyunsoo Kim, Seung Yoon, Soon-Wan Kwon, Yongmin Ju, Minje Kim, Wooseok Yi, Shinhee Han, Baeseong Kwon, Boyoung Seo, Kilho Lee, Gwan-Hyeob Koh, Kangho Lee, Yoonjong Song, Changkyu Choi, Donhee Ham, and Sang Kim (Jan. 2022). "A crossbar array of magnetoresistive memory devices for in-memory computing". In: *Nature* 601, pp. 211–216. DOI: 10.1038/s41586-021-04196-6.

Kaiser, J., A. Friedrich, J. C. V. Tieck, D. Reichard, A. Roennau, E. Neftci, and R. Dillmann (Nov. 2020a). "Embodied Neuromorphic Vision with Continuous Random Backpropagation". In: *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*. 2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob). ISSN: 2155-1782, pp. 1202–1209. DOI: 10.1109/BioRob49111.2020.9224330.

Kaiser, Jacques, Hesham Mostafa, and Emre Neftci (2020b). "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)". In: *Frontiers in Neuroscience* 14. ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00424.

Karpov, I. V., M. Mitra, D. Kau, G. Spadini, Y. A. Kryukov, and V. G. Karpov (2007). "Fundamental drift of parameters in chalcogenide phase change memory". In: *Journal of Applied Physics* 102.12, p. 124503. DOI: 10.1063/1.2825650.

Kasabov, Nikola K. (Apr. 2014). "NeuCube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data". eng. In: *Neural Networks: The Official Journal of the International Neural Network Society* 52, pp. 62–76. ISSN: 1879-2782. DOI: 10.1016/j.neunet.2014.01.006.

Khacef, Lyes, Nassim Abderrahmane, and Benoît Miramond (2018). "Confronting machine-learning with neuroscience for neuromorphic architectures design". In: *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489241.

Kheradpisheh, Saeed Reza, Mohammad Ganjtabesh, Simon J. Thorpe, and Timothée Masquelier (Mar. 2018). "STDP-based spiking deep convolutional neural networks for object recognition". In: *Neural Networks* 99, pp. 56–67. ISSN: 08936080. DOI: 10.1016/j.neunet.2017.12.005. arXiv: 1611.01421. URL: http://arxiv.org/abs/1611.01421 (visited on 09/11/2020).

Kheradpisheh, Saeed Reza and Timothée Masquelier (June 2020). "S4NN: temporal backpropagation for spiking neural networks with one spike per neuron". In: *Int. J. Neur. Syst.* 30.6, p. 2050027. ISSN: 0129-0657, 1793-6462. DOI: 10.1142/S0129065720500276. arXiv: 1910.09495. URL: http://arxiv.org/abs/1910.09495 (visited on 09/17/2020).

Kim, Kwantae, Chang Gao, Rui Graça, Ilya Kiselev, Hoi-Jun Yoo, Tobi Delbruck, and Shih-Chii Liu (2022a). "A 23μW Solar-Powered Keyword-Spotting ASIC with Ring-Oscillator-Based Time-Domain Feature Extraction". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65, pp. 1–3. DOI: 10.1109/ISSCC42614.2022.9731708.

Kim, Youngeun and Priyadarshini Panda (Nov. 2020). "Revisiting Batch Normalization for Training Low-latency Deep Spiking Neural Networks from Scratch". en. In: *arXiv:2010.01729*. (Visited on 07/22/2021).

Kim, Youngeun, Hyoungseob Park, Abhishek Moitra, Abhiroop Bhattacharjee, Yeshwanth Venkatesha, and Priyadarshini Panda (2022b). "Rate Coding Or Direct Coding: Which One Is Better For Accurate, Robust, And Energy-Efficient Spiking Neural Networks?" In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 71–75. DOI: 10.1109/ICASSP43922.2022.9747906.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: DOI: 10.48550/arxiv.1412.6980.

Krizhevsky, Alex and Geoffrey Hinton (2009). "Learning multiple layers of features from tiny images". In: 0.

Kumarasinghe, Kaushalya, Nikola Kasabov, and Denise Taylor (Dec. 2021). "Brain-inspired spiking neural networks for decoding and understanding muscle activity and kinematics from electroencephalography signals during hand movements". en. In: *Scientific Reports* 11.1, p. 2486. ISSN: 2045-2322. DOI: 10.1038/s41598-021-81805-4. (Visited on 06/03/2021).

Lapicque, L. (1907). "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization". In: *Journal of Physiol Pathol Générale* 9, pp. 620–635.

Le, Quoc V., Navdeep Jaitly, and Geoffrey E. Hinton (2015). "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units". In: DOI: 10.48550/ARXIV.1504.00941.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

Ledinauskas, Eimantas, Julius Ruseckas, Alfonsas Juršėnas, and Giedrius Buračas (June 2020). "Training Deep Spiking Neural Networks". In: *arXiv:2006.04436 [cs]*. URL: http://arxiv.org/abs/2006.04436 (visited on 12/08/2020).

Lee, Chankyu, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy (2018). "Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning". In: *Front. Neurosci.* 12. Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00435. URL: https://www.frontiersin.org/articles/10.3389/fnins.2018.00435/full (visited on 09/21/2020).

Lee, Chankyu, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy (Feb. 2020a). "Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures". en. In: *Frontiers in Neuroscience* 14, p. 119. ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00119. (Visited on 09/08/2020).

Lee, Hunjun, Chanmyeong Kim, Seungho Lee, Eunjin Baek, and Jangwoo Kim (Sept. 2021a). "An accurate and fair evaluation methodology for SNN-based inferencing with full-stack hardware design space explorations". en. In: *Neurocomputing* 455,

pp. 125–138. ISSN: 09252312. DOI: 10.1016/j.neucom.2021.05.020. (Visited on 11/09/2021).

Lee, Jeong-Jun, Jianhao Chen, Wenrui Zhang, and Peng Li (July 2021b). "Systolic-Array Spiking Neural Accelerators with Dynamic Heterogeneous Voltage Regulation". en. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. Shenzhen, China: IEEE, pp. 1–7. ISBN: 978-1-66543-900-8. DOI: 10.1109/IJCNN52387.2021.9534037. (Visited on 11/09/2021).

Lee, Jeong-Jun and Peng Li (Oct. 2020b). "Reconfigurable Dataflow Optimization for Spatiotemporal Spiking Neural Computation on Systolic Array Accelerators". en. In: *2020 IEEE 38th International Conference on Computer Design (ICCD)*. Hartford, CT, USA: IEEE, pp. 57–64. ISBN: 978-1-72819-710-4. DOI: 10.1109/ICCD50377.2020.00027. (Visited on 11/09/2021).

Lee, Jeongjun, Renqian Zhang, Wenrui Zhang, Yu Liu, and Peng Li (2020c). "Spike-Train Level Direct Feedback Alignment: Sidestepping Backpropagation for On-Chip Training of Spiking Neural Nets". In: *Front. Neurosci.* 14. Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00143. URL: https://www.frontiersin.org/articles/10.3389/fnins.2020.00143/full (visited on 10/01/2020).

Lee, Jun Haeng, Tobi Delbruck, and Michael Pfeiffer (2016). "Training Deep Spiking Neural Networks Using Backpropagation". English. In: *Frontiers in Neuroscience* 10. ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508. (Visited on 09/10/2020).

Lemaire, Edgar, Loîc Cordone, Andrea Castagnetti, Pierre-Emmanuel Novac, Jonathan Courtois, and Benoit Miramond (Nov. 2022). "An Analytical Estimation of Spiking Neural Networks Energy Efficiency". In: *International Conference on Neural Information Processing (ICONIP)*. Ed. by Springer. Vol. 13623. ITT Indore, India, p. 8. DOI: 10.1007/978-3-031-30105-6_48. URL: https://hal.science/hal-03875214.

Leonard, R Gary and George Doddington (1993). "Tidigits speech corpus". In: *Texas Instruments, Inc*. DOI: 10.35111/72xz-6x59.

Li, Chen, Runze Chen, Christoforos Moutafis, and Steve Furber (2020). "Robustness to Noisy Synaptic Weights in Spiking Neural Networks". In: *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207019.

Li, Chen, Lei Ma, and Steve Furber (2022). "Quantization Framework for Fast Spiking Neural Networks". In: *Frontiers in Neuroscience* 16. ISSN: 1662-453X. DOI: 10.3389/fnins.2022.918793.

Li, Da, Xinbo Chen, Michela Becchi, and Ziliang Zong (2016). "Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs". In: *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pp. 477–484. DOI: 10.1109/BDCloud-SocialCom-SustainCom.2016.76.

Li, Hongmin, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi (2017). "CIFAR10-DVS: An Event-Stream Dataset for Object Classification". In: *Front. Neurosci.* 11. Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00309. URL: https://www.frontiersin.org/articles/10.3389/fnins.2017.00309/full (visited on 06/03/2021).

Li, Yuhang, Shikuang Deng, Xin Dong, Ruihao Gong, and Shi Gu (2021). "A Free Lunch From ANN: Towards Efficient, Accurate Spiking Neural Networks Calibration". In:

Proceedings of the 38th International Conference on International Conference on Machine Learning - Volume 139, pp. 6316–6325.

Liao, Qianli, Joel Z Leibo, and Tomaso Poggio (2016). "How Important Is Weight Symmetry in Backpropagation?" In: p. 10.

Lichtsteiner, Patrick, Christoph Posch, and Tobi Delbruck (Feb. 2008). *A 128-128 120 dB 15 -s Latency Asynchronous Temporal Contrast Vision Sensor | IEEE Journals & Magazine | IEEE Xplore*. URL: https://ieeexplore.ieee.org/document/4444573/?reason=concurrency (visited on 06/03/2021).

Lillicrap, Timothy P., Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman (Dec. 2016). "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nat Commun* 7.1, p. 13276. ISSN: 2041-1723. DOI: 10.1038/ncomms13276. URL: http://www.nature.com/articles/ncomms13276 (visited on 10/15/2020).

Lopez-Espejo, Ivan, Zheng-Hua Tan, John H. L. Hansen, and Jesper Jensen (2022). "Deep Spoken Keyword Spotting: An Overview". In: *IEEE Access* 10, pp. 4169–4199. DOI: 10.1109/ACCESS.2021.3139508.

Lotfi Rezaabad, Ali and Sriram Vishwanath (2020). "Long Short-Term Memory Spiking Networks and Their Applications". In: *International Conference on Neuromorphic Systems 2020*. ICONS 2020: International Conference on Neuromorphic Systems 2020. Oak Ridge TN USA: ACM, pp. 1–9. ISBN: 978-1-4503-8851-1. DOI: 10.1145/3407197.3407211. (Visited on 10/26/2020).

Malekzadeh, Elaheh, Nezam Rohbani, Zhonghai Lu, and Masoumeh Ebrahimi (2021). "The Impact of Faults on DNNs: A Case Study". In: *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 1–6. DOI: 10.1109/DFT52944.2021.9568340.

Mao, Shitong and Ervin Sejdić (2022). "A Review of Recurrent Neural Network-Based Methods in Computational Physiology". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21. DOI: 10.1109/TNNLS.2022.3145365.

Mead, C. (1990). "Neuromorphic electronic systems". In: *Proceedings of the IEEE* 78.10, pp. 1629–1636. DOI: 10.1109/5.58356.

Merolla, Paul A., John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha (2014). "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197, pp. 668–673. ISSN: 0036-8075. DOI: 10.1126/science.1254642.

Minguet Lopez, J., T. Hirtzlin, M. Dampfhoffer, L. Grenouillet, L. Reganaz, G. Navarro, C. Carabasse, E. Vianello, T. Magis, D. Deleruyelle, M. Bocquet, J. M. Portal, F. Andrieu, and G. Molas (Dec. 2021). "OxRAM + OTS optimization for binarized neural network hardware implementation". In: *Semiconductor Science and Technology* 37.1. Publisher: IOP Publishing, p. 014001. DOI: 10.1088/1361-6641/ac31e2. URL: https://dx.doi.org/10.1088/1361-6641/ac31e2.

Minguet Lopez, Joel, Manon Dampfhoffer, Tifenn Hirtzlin, Lucas Reganaz, Laurent Grenouillet, Gabriele Navarro, Mathieu Bernard, Thomas Magis, Catherine Carabasse, Niccolo Castellani, Valentina Meli, Elisa Vianello, Damien Deleruyelle, Jean-Michel Portal, Gabriel Molas, and François Andrieu (2023). "1S1R Sub-Threshold Operation in Crossbar Arrays for Neural Networks Hardware Implementation". In: *2023*

30th International Conference on Mixed Design of Integrated Circuits and System (MIXDES), pp. 1–6. DOI: 10.23919/MIXDES58562.2023.10203226.

Minguet Lopez, Joel, Quentin Rafhay, Manon Dampfhoffer, Lucas Reganaz, Niccolo Castellani, Valentina Meli, Simon Martin, Laurent Grenouillet, Gabriele Navarro, Thomas Magis, Catherine Carabasse, Tifenn Hirtzlin, Elisa Vianello, Damien Deleruyelle, Jean-Michel Portal, Gabriel Molas, and François Andrieu (2022). "1S1R Optimization for High-Frequency Inference on Binarized Spiking Neural Networks". In: *Advanced Electronic Materials*, p. 2200323. DOI: 10.1002/aelm.202200323.

Mishra, Asit, Eriko Nurvitadhi, Jeffrey J. Cook, and Debbie Marr (Sept. 2017). "WRPN: Wide Reduced-Precision Networks". In: arXiv: 1709.01134. URL: http://arxiv.org/abs/1709.01134 (visited on 08/27/2021).

Mittal, Sparsh and Sumanth Umesh (2021). "A survey On hardware accelerators and optimization techniques for RNNs". In: *Journal of Systems Architecture* 112, p. 101839. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2020.101839.

Moon, Suhong, Kwanghyun Shin, and Dongsuk Jeon (2019). "Enhancing Reliability of Analog Neural Network Processors". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.6, pp. 1455–1459. DOI: 10.1109/TVLSI.2019.2893256.

Moradi, Saber, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri (Feb. 2018). "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)". en. In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1, pp. 106–122. ISSN: 1932-4545, 1940-9990. DOI: 10.1109/TBCAS.2017.2759700. (Visited on 08/03/2021).

Mostafa, H., B. U. Pedroni, S. Sheik, and G. Cauwenberghs (May 2017). "Fast classification using sparsely active spiking networks". In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. ISSN: 2379-447X, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050527.

Mostafa, Hesham (2018). "Supervised Learning Based on Temporal Coding in Spiking Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.7, pp. 3227–3235. DOI: 10.1109/TNNLS.2017.2726060.

Mostafa, Hesham, Vishwajith Ramesh, and Gert Cauwenberghs (2018). "Deep Supervised Learning Using Local Errors". English. In: *Frontiers in Neuroscience* 12. Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00608.

Mozafari, Milad, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, and Timothée Masquelier (2019a). "SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks With at Most One Spike per Neuron". In: *Frontiers in Neuroscience* 13. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00625. URL: https://www.frontiersin.org/articles/10.3389/fnins.2019.00625.

Mozafari, Milad, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J. Thorpe, and Timothée Masquelier (Oct. 1, 2019b). "Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks". In: *Pattern Recognition* 94, pp. 87–95. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2019.05.015. URL: http://www.sciencedirect.com/science/article/pii/S0031320319301906 (visited on 10/14/2020).

Mozafari, Milad, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh (Dec. 2018). "First-Spike-Based Visual Categorization Using Reward-Modulated STDP". In: *IEEE Transactions on Neural Networks*

and Learning Systems 29.12. IEEE Transactions on Neural Networks and Learning Systems, pp. 6178–6190. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2018.2826721.

Murray, A.F. and P.J. Edwards (1994). "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training". In: *IEEE Transactions on Neural Networks* 5.5, pp. 792–802. DOI: 10.1109/72.317730.

Narayanan, Surya, Karl Taht, Rajeev Balasubramonian, Edouard Giacomin, and Pierre-Emmanuel Gaillardon (May 2020). "SpinalFlow: An Architecture and Dataflow Tailored for Spiking Neural Networks". en. In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, pp. 349–362. ISBN: 978-1-72814-661-4. DOI: 10.1109/ISCA45697.2020.00038. (Visited on 11/09/2021).

Nawrocki, Robert A., Richard M. Voyles, and Sean E. Shaheen (2016). "A Mini Review of Neuromorphic Architectures and Implementations". In: *IEEE Transactions on Electron Devices* 63.10, pp. 3819–3829. DOI: 10.1109/TED.2016.2598413.

Neftci, Emre, Hesham Mostafa, and Friedemann Zenke (Nov. 2019). "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks". In: *IEEE Signal Processing Magazine* 36, pp. 51–63. DOI: 10.1109/MSP.2019.2931595.

Neftci, Emre O., Charles Augustine, Somnath Paul, and Georgios Detorakis (2017). "Event-Driven Random Back-Propagation: Enabling Neuromorphic Deep Learning Machines". In: *Front. Neurosci.* 11. Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00324. URL: https://www.frontiersin.org/articles/10.3389/fnins.2017.00324/full (visited on 09/30/2020).

Nirschl, T., J.B. Philipp, T.D. Happ, G.W. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y.C. Chen, Y. Zhu, R. Bergmann, H.-L. Lung, and C. Lam (2007). "Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory". In: *2007 IEEE International Electron Devices Meeting*, pp. 461–464. DOI: 10.1109/IEDM.2007.4418973.

Nøkland, Arild (2016). "Direct Feedback Alignment Provides Learning in Deep Neural Networks". In: p. 9.

Orchard, Garrick, E. Paxon Frady, Daniel Ben Dayan Rubin, Sophia Sanborn, Sumit Bam Shrestha, Friedrich T. Sommer, and Mike Davies (2021). "Efficient Neuromorphic Signal Processing with Loihi 2". In: *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 254–259. DOI: 10.1109/SiPS52927.2021.00053.

Orchard, Garrick, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor (2015). "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades". In: *Front. Neurosci.* 9. Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437. URL: https://www.frontiersin.org/articles/10.3389/fnins.2015.00437/full (visited on 06/03/2021).

Panda, Priyadarshini, Sai Aparna Aketi, and Kaushik Roy (2020). "Toward Scalable, Efficient, and Accurate Deep Spiking Neural Networks With Backward Residual Connections, Stochastic Softmax, and Hybridization". In: *Frontiers in Neuroscience* 14. ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00653. (Visited on 04/13/2021).

Park, Daniel S., William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le (2019). "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Proceedings of Interspeech*.

Park, S., S. Kim, B. Na, and S. Yoon (July 2020). "T2FSNN: Deep Spiking Neural Networks with Time-to-first-spike Coding". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. ISSN: 0738-100X, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218689.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc.

Pehle, Christian, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel (2022). "The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity". In: *Frontiers in Neuroscience* 16. ISSN: 1662-453X. DOI: 10.3389/fnins.2022.795876. URL: https://www.frontiersin.org/articles/10.3389/fnins.2022.795876.

Pellegrini, Thomas, Romain Zimmer, and Timothée Masquelier (Jan. 2021). "Low-activity supervised convolutional spiking neural networks applied to speech commands recognition". In: *IEEE Spoken Language Technology Workshop 2021*. Proc. 2021 IEEE Spoken Language Technology Workshop (SLT). IEEE Xplore, pp. 97–103. DOI: 10.1109/SLT48900.2021.9383587.

Perez-Nieves, Nicolas, Vincent C. H. Leung, Pier Luigi Dragotti, and Dan F. M. Goodman (2021). "Neural heterogeneity promotes robust learning". In: *Nature Communications* 12.1, p. 5791. ISSN: 2041-1723. DOI: 10.1038/s41467-021-26022-3.

Ponghiran, Wachirawit and Kaushik Roy (2021). "Hybrid Analog-Spiking Long Short-Term Memory for Energy Efficient Computing on Edge Devices". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 581–586. DOI: 10.23919/DATE51398.2021.9473953.

— (2022). "Spiking Neural Networks with Improved Inherent Recurrence Dynamics for Sequential Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36, pp. 8001–8008.

Putra, Rachmad Vidya Wicaksana, Muhammad Abdullah Hanif, and Muhammad Shafique (Apr. 2021). "ROMANet: Fine-Grained Reuse-Driven Off-Chip Memory Access Management and Data Organization for Deep Neural Network Accelerators". en. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.4, pp. 702–715. ISSN: 1063-8210, 1557-9999. DOI: 10.1109/TVLSI.2021.3060509.

Pytorch torchvision models (2021). URL: https://pytorch.org/vision/stable/models.html.

Rathi, Nitin, Amogh Agrawal, Chankyu Lee, Adarsh Kumar Kosta, and Kaushik Roy (2021a). "Exploring Spike-Based Learning for Neuromorphic Computing: Prospects and Perspectives". In: *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 902–907. DOI: 10.23919/DATE51398.2021.9473964.

Rathi, Nitin and Kaushik Roy (2021b). "DIET-SNN: A Low-Latency Spiking Neural Network With Direct Input Encoding and Leakage and Threshold Optimization". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–9. DOI: 10.1109/TNNLS.2021.3111897.

Rathi, Nitin, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy (2020). "Enabling Deep Spiking Neural Networks with Hybrid Conversion and Spike Timing Dependent Backpropagation". In: *arXiv:2005.01807 [cs, stat]*. URL: http://arxiv.org/abs/2005.01807.

Ravanelli, M., P. Brakel, M. Omologo, and Y. Bengio (2018). "Light Gated Recurrent Units for Speech Recognition". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.2, pp. 92–102. ISSN: 2471-285X. DOI: 10.1109/TETCI.2017.2762739.

Reganaz, L., D. Deleruyelle, Q. Rafhay, J. Minguet Lopez, N. Castellani, J. F. Nodin, A. Bricalli, G. Piccolboni, G. Molas, and F. Andrieu (n.d.). "Investigation of resistance fluctuations in ReRAM: physical origin, temporal dependence and impact on memory reliability". In: *accepted to 2023 IEEE International Reliability Physics Symposium (IRPS)*.

Roy, Arnab, Swagath Venkataramani, Neel Gala, Sanchari Sen, Kamakoti Veezhinathan, and Anand Raghunathan (2017). "A Programmable Event-driven Architecture for Evaluating Spiking Neural Networks". In: *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6. DOI: 10.1109/ISLPED.2017.8009176.

Roy, Kaushik, Akhilesh Jaiswal, and Priyadarshini Panda (Nov. 2019). "Towards spike-based machine intelligence with neuromorphic computing". In: *Nature* 575.7784, pp. 607–617. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1677-2.

Rueckauer, B. and S. Liu (May 2018). "Conversion of analog to spiking neural networks using sparse temporal coding". In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. DOI: 10.1109/ISCAS.2018.8351295.

Rueckauer, Bodo, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu (2017). "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification". In: *Frontiers in Neuroscience* 11, p. 682. ISSN: 1662-4548. DOI: 10.3389/fnins.2017.00682.

Sakemi, Yusuke, Kai Morino, Takashi Morie, and Kazuyuki Aihara (2023). "A Supervised Learning Algorithm for Multilayer Spiking Neural Networks Based on Temporal Coding Toward Energy-Efficient VLSI Processor Design". In: *IEEE Transactions on Neural Networks and Learning Systems* 34.1, pp. 394–408. DOI: 10.1109/TNNLS.2021.3095068.

Sengupta, Abhronil, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy (2019). "Going Deeper in Spiking Neural Networks: VGG and Residual Architectures". In: *Frontiers in Neuroscience* 13, p. 95. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00095.

Shini, R. Subha and V.D. Ambeth Kumar (2021). "Recurrent Neural Network based Text Summarization Techniques by Word Sequence Generation". In: *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pp. 1224–1229. DOI: 10.1109/ICICT50816.2021.9358764.

Shrestha, Amar, Khadeer Ahmed, Yanzhi Wang, David P. Widemann, Adam T. Moody, Brian C. Van Essen, and Qinru Qiu (2017). "A spike-based long short-term memory on a neurosynaptic processor". In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 631–637. DOI: 10.1109/ICCAD.2017.8203836.

Shrestha, Sumit Bam and Garrick Orchard (2018). "SLAYER: Spike Layer Error Reassignment in Time". In: *Advances in Neural Information Processing Systems*. Vol. 31.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (Jan. 28, 2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489. ISSN: 0028-0836, 1476-4687.

Solinas., M., S. Rousset., R. Cohendet., Y. Bourrier., M. Mainsant., A. Molnos., M. Reyboz., and M. Mermillod. (2021). "Beneficial Effect of Combined Replay for Continual Learning". In: *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*. INSTICC. SciTePress, pp. 205–217. ISBN: 978-989-758-484-8. DOI: 10.5220/0010251202050217.

Srinivasan, Gopalakrishnan and Kaushik Roy (Feb. 11, 2019). "ReStoCNet: Residual Stochastic Binary Convolutional Spiking Neural Network for Memory-Efficient Neuromorphic Computing". In: *arXiv:1902.04161 [cs]*. arXiv: 1902.04161. URL: http://arxiv.org/abs/1902.04161 (visited on 10/14/2020).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (Jan. 2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1, 1929–1958. ISSN: 1532-4435.

Stanojevic, Ana, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini, Angeliki Pantazi, and Wulfram Gerstner (2022). *An Exact Mapping From ReLU Networks to Spiking Neural Networks*. arXiv: 2212.12522 [cs.NE].

Stimberg, Marcel, Romain Brette, and Dan FM Goodman (Aug. 2019). "Brian 2, an intuitive and efficient neural simulator". In: *eLife* 8. Ed. by Frances K Skinner, e47314. ISSN: 2050-084X. DOI: 10.7554/eLife.47314.

Stöckl, Christoph and Wolfgang Maass (Mar. 2021). "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes". In: *Nature Machine Intelligence* 3. DOI: 10.1038/s42256-021-00311-4.

Sze, Vivienne, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer (Aug. 2017). "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *arXiv:1703.09039 [cs]*. arXiv: 1703.09039. URL: http://arxiv.org/abs/1703.09039 (visited on 09/25/2020).

Sze, Vivienne, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer (June 2020). "Efficient Processing of Deep Neural Networks". en. In: *Synthesis Lectures on Computer Architecture* 15.2, pp. 1–341. ISSN: 1935-3235, 1935-3243. DOI: 10.2200/S01004ED1V01Y202004CAC050. (Visited on 11/10/2021).

Tavanaei, Amirhossein and Anthony Maida (2019). "BP-STDP: Approximating backpropagation using spike timing dependent plasticity". In: *Neurocomputing* 330, pp. 39–47. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.11.014.

Thiele, Johannes C., Olivier Bichler, and Antoine Dupret (June 14, 2018). "Event-Based, Timescale Invariant Unsupervised Online Deep Learning With STDP". In: *Front. Comput. Neurosci.* 12, p. 46. ISSN: 1662-5188. DOI: 10.3389/fncom.2018.00046. URL: https://www.frontiersin.org/article/10.3389/fncom.2018.00046/full (visited on 09/08/2020).

Thiele, Johannes Christian, Olivier Bichler, and Antoine Dupret (June 2019). "SpikeGrad: An ANN-equivalent Computation Model for Implementing Backpropagation with Spikes". en. In: *arXiv:1906.00851 [cs]*. (Visited on 09/08/2020).

Valentian, A., F. Rummens, E. Vianello, T. Mesquida, C. L. de Boissac, O. Bichler, and C. Reita (Dec. 2019). "Fully Integrated Spiking Neural Network with Analog Neurons and RRAM Synapses". In: *2019 IEEE International Electron Devices Meeting (IEDM)*. 2019 IEEE International Electron Devices Meeting (IEDM). ISSN: 2156-017X, pp. 14.3.1–14.3.4. DOI: 10.1109/IEDM19573.2019.8993431.

Vatajelu, Elena-Ioana, Giorgio Di Natale, and Lorena Anghel (2019). "Special Session: Reliability of Hardware-Implemented Spiking Neural Networks (SNN)". In: *2019 IEEE 37th VLSI Test Symposium (VTS)*, pp. 1–8. DOI: 10.1109/VTS.2019.8758653.

Viale, Alberto, Alberto Marchisio, Maurizio Martina, Guido Masera, and Muhammad Shafique (2021). "CarSNN: An Efficient Spiking Neural Network for Event-Based Autonomous Cars on the Loihi Neuromorphic Research Processor". In: arXiv: 2107.00401. (Visited on 08/03/2021).

Wan, Weier, Rajkumar Kubendran, Clemens J. S. Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen R. Deiss, Priyanka Raina, He Qian, Bin Gao, Siddharth Joshi, Huaqiang Wu, H.-S. Philip Wong, and Gert Cauwenberghs (2022). "A compute-in-memory chip based on resistive random-access memory". In: *Nat.* 608.7923, pp. 504–512. DOI: 10.1038/s41586-022-04992-8.

Wang, Dewei, Sung Justin Kim, Minhao Yang, Aurel A. Lazar, and Mingoo Seok (2021). "9.9 A Background-Noise and Process-Variation-Tolerant 109nW Acoustic Feature Extractor Based on Spike-Domain Divisive-Energy Normalization for an Always-On Keyword Spotting Device". In: *2021 IEEE International Solid- State Circuits Conference (ISSCC)*. Vol. 64, pp. 160–162. DOI: 10.1109/ISSCC42613.2021.9365969.

Wang, Jiadong, Jibin Wu, Malu Zhang, Qi Liu, and Haizhou Li (2022). "A Hybrid Learning Framework for Deep Spiking Neural Networks with One-Spike Temporal Coding". In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8942–8946. DOI: 10.1109/ICASSP43922.2022.9746792.

Warden, Pete (2018). *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. DOI: 10.48550/ARXIV.1804.03209. URL: https://arxiv.org/abs/1804.03209.

Wu, Jibin, Yansong Chua, Malu Zhang, Guoqi Li, Haizhou Li, and Kay Chen Tan (2021). "A Tandem Learning Rule for Effective Training and Rapid Inference of Deep Spiking Neural Networks". en. In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2021.3095724. (Visited on 11/10/2021).

Wu, Jibin, Yansong Chua, Malu Zhang, Qu Yang, Guoqi Li, and Haizhou Li (July 2019a). "Deep Spiking Neural Network with Spike Count based Learning Rule". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019 International Joint Conference on Neural Networks (IJCNN). Budapest, Hungary: IEEE, pp. 1–6. ISBN: 978-1-72811-985-4. DOI: 10.1109/IJCNN.2019.8852380. (Visited on 09/08/2020).

Wu, Jibin, Emre Yılmaz, Malu Zhang, Haizhou Li, and Kay Tan (Mar. 2020). "Deep Spiking Neural Networks for Large Vocabulary Automatic Speech Recognition". In: *Frontiers in Neuroscience* 14. DOI: 10.3389/fnins.2020.00199.

Wu, Yujie, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi (May 2018). "Spatio-Temporal Backpropagation for Training High-performance Spiking Neural Networks". en. In: *Frontiers in Neuroscience* 12, p. 331. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00331. (Visited on 09/08/2020).

Wu, Yujie, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi (July 2019b). "Direct Training for Spiking Neural Networks: Faster, Larger, Better". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.1, pp. 1311–1318. DOI: 10.1609/aaai.v33i01.33011311.

Yan, Zheyu, Xiaobo Sharon Hu, and Yiyu Shi (2023). "On the Reliability of Computing-in-Memory Accelerators for Deep Neural Networks". In: *System Dependability and Analytics: Approaching System Dependability from Data, System and Analytics Perspectives*. Cham, pp. 167–190. ISBN: 978-3-031-02063-6. DOI: 10.1007/978-3-031-02063-6_9.

Yang, Tien-Ju and Vivienne Sze (2019a). "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators". In: *2019 IEEE International Electron Devices Meeting (IEDM)*, pp. 22.1.1–22.1.4. DOI: 10.1109/IEDM19573.2019.8993662.

— (2019b). "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators". In: *2019 IEEE International Electron Devices Meeting (IEDM)*, pp. 22.1.1–22.1.4. DOI: 10.1109/IEDM19573.2019.8993662.

Yao, P, Wu H, Gao B, Tang J, Zhang Q, Zhang W, Yang JJ, and Qian H (2020). "Fully hardware-implemented memristor convolutional neural network". In: *Nature* 577, pp. 641–646. DOI: 10.1038/s41586-020-1942-4.

Yin, Bojian, Federico Corradi, and Sander M. Bohté (2020). "Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks". In: *International Conference on Neuromorphic Systems 2020*. ICONS 2020: International Conference on Neuromorphic Systems 2020. Oak Ridge TN USA: ACM, pp. 1–8. ISBN: 978-1-4503-8851-1. DOI: 10.1145/3407197.3407225. (Visited on 07/22/2021).

— (2021). "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks". In: *Nature Machine Intelligence* 3.10, pp. 905–913. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00397-w.

Zenke, Friedemann and Surya Ganguli (2018). "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks". In: *Neural Computation* 30.6, pp. 1514–1541. DOI: 10.1162/neco_a_01086.

Zenke, Friedemann and Tim P. Vogels (Mar. 2021). "The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks". In: *Neural Computation* 33.4, pp. 899–925. ISSN: 0899-7667. DOI: 10.1162/neco_a_01367.

Zhang, Duzhen, Tielin Zhang, Shuncheng Jia, Qing Wang, and Bo Xu (2022). "Recent Advances and New Frontiers in Spiking Neural Networks". In: *International Joint Conference on Artificial Intelligence*.

Zhang, Malu, Jiadong Wang, Zhixuan Zhang, Ammar Belatreche, Jibin Wu, Yansong Chua, Hong Qu, and Haizhou Li (Mar. 2020). "Spike-Timing-Dependent Back Propagation in Deep Spiking Neural Networks". In: *arXiv:2003.11837 [cs]*. arXiv: 2003.11837. URL: http://arxiv.org/abs/2003.11837 (visited on 09/17/2020).

Zhang, Wenrui and Peng Li (2020). "Temporal Spike Sequence Learning via Backpropagation for Deep Spiking Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 12022–12033.

Zhang, Yundong, Naveen Suda, Liangzhen Lai, and Vikas Chandra (2017). *Hello Edge: Keyword Spotting on Microcontrollers*. DOI: 10.48550/ARXIV.1711.07128. URL: https://arxiv.org/abs/1711.07128.

Zhao, Meiran, Huaqiang Wu, Bin Gao, Qingtian Zhang, Wei Wu, Shan Wang, Yue Xi, Dong Wu, Ning Deng, Shimeng Yu, Hong-Yu Chen, and He Qian (2017). "Investigation of statistical retention of filamentary analog RRAM for neuromophic computing". In: *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 39.4.1–39.4.4. DOI: 10.1109/IEDM.2017.8268522.

Zheng, Hanle, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li (May 2021). "Going Deeper With Directly-Trained Larger Spiking Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12, pp. 11062–11070.

Zhou, Shibo, Ying Chen, Xiaohua Li, and Arindam Sanyal (2020). "Deep SCNN-Based Real-Time Object Detection for Self-Driving Vehicles Using LiDAR Temporal Data". In: *IEEE Access* 8, pp. 76903–76912. DOI: 10.1109/ACCESS.2020.2990416.

Zhou, Shibo, Xiaohua Li, Ying Chen, Sanjeev T. Chandrasekaran, and Arindam Sanyal (May 2021). "Temporal-Coded Deep Spiking Neural Network with Easy Training and Robust Performance". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12. Number: 12, pp. 11143–11151. ISSN: 2374-3468. (Visited on 11/10/2021).