الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique
Département d'Électronique

End of studies project thesis

Submited for the fulfilment of state engineer degree in Electronic

---

Spiking Neural Networks Optimization

Realized by :
FERHAT Hiba El Batoul

Supervised by :
Dr. ESHAGHIAN Jason (UWA, UMich)
Dr. BERRANI Sid-Ahmed (ENP)

Defended on June 30th, 2022, before the jury composed of :

M. BELOUCHRANI Adel	:	ENP	- President
M. ADNANE Mourad	:	ENP	- Examiner
M. BERRANI Sid-Ahmed	:	ENP	- Supervisor

ENP 2022

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique
Département d'Électronique

End of studies project thesis

Submited for the fulfilment of state engineer degree in Electronic

---

Spiking Neural Networks Optimization

Realized by :
FERHAT Hiba El Batoul

Supervised by :
Dr. ESHAGHIAN Jason (UWA, UMich)
Dr. BERRANI Sid-Ahmed (ENP)

Defended on June 30th, 2022, before the jury composed of :

M. BELOUCHRANI Adel	:	ENP	- President
M. ADNANE Mourad	:	ENP	- Examiner
M. BERRANI Sid-Ahmed	:	ENP	- Supervisor

ENP 2022

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique
Département d'Électronique

Projet de Fin d'Études

Pour l'obtention du diplôme dingénieur détat en Electronique

Optimization des réseaux de neurones a impulssions

Réaliser par :
FERHAT Hiba El Batoul

Sous la direction de :
Dr. ESHAGHIAN Jason (UWA, UMich)
Dr. BERRANI Sid-Ahmed (ENP)

Présenté et soutenu publiquement le 30 Juin 2022, devant le jury :

M. BELOUCHRANI Adel	:	ENP	- President
M. ADNANE Mourad	:	ENP	- Examinateur
M. BERRANI Sid-Ahmed	:	ENP	- Encadrant

ENP 2022

DEDICATION

À mes chers parents,
À mes frères Hichem, Mehdi, Chouaib, Tarek et ma soeur Yousra
À mon oncle Abdellah,
À ma famille,
À mes amis Lyna, Ouiem et Thanina,
À tous ceux qui me sont chers, à vous tous.

Merci pour votre soutien.

- Hiba El Batoul

ACKNOWLEDGEMENT

I would like to take the opportunity to express my deepest gratitude and thanks to my supervisors Dr. Jason K. Eshaghian and Dr. Berrani Sid-Ahmed, for their constant guidance, assistance, availability, and their invaluable support as well as all the knowledge shared during the course of this work.

A special thanks to Professor Mohammed Bennamoun, Professor at University of Western Australia, for the encouragement, support and giving me the chance to engage in cutting edge research.

A special thanks to all electronics department professors for inspiring me, encouraging me, and pushing my limits to surpass myself. I am very grateful for this exceptional journey!

I would like to thank the committee members for reviewing our dissertation and for their constructive analysis and insightful comments of the present work.

ملخص - الشبكات العصبية النابضة هي أساليب جديدة لحل مشاكل كفاءة الذكاء الاصطناعي التقليدي. ومع ذلك ، هناك فجوة كبيرة في الأداء بين تحفيز الشبكات العصبية والشبكات العصبية الاصطناعية. يساهم هذا العمل في تحسين الشبكات العصبية المتصاعدة ، من أجل تقليل هذه الفجوة. تتعلق الطرق المقترحة بتحسين البنية وإنشاء وظائف خسارة جديدة ، من أجل تجنب مشكلة الخلايا العصبية الميتة. يتم تنفيذها باستخدام مكتبة hcroTyP hcroTnnS ويتم تقييمها باستخدام مجموعتي بيانات مختلفتين. بفضل هذا العمل ، تم فتح آفاق جديدة للتطبيقات ، بالإضافة إلى العديد من تقنيات التحسين الممكنة.

كلمات مفتاحية : .الانتشار العكسي, خوارزمية النسب المتدرج, الحوسبة العصبية

Résumé - Les réseaux de neurones à impulsions offrent une nouvelle approche pour résoudre les problèmes d'efficacité de l'intelligence artificielle (IA) conventionnelle, mais il existe un écart de performance important entre les réseaux de neurones à impulsions et les réseaux de neurones artificiels. Ce travail contribue à l'optimisation des réseaux de neurones à impulsions afin de réduire cet écart. Les méthodes proposées sont axées sur l'optimisation de l'architecture et le développement de nouvelles fonctions de perte basées sur les pics, afin d'éviter le problème des neurones morts. Les techniques proposées sont mises en uvre à l'aide des cadres PyTorch et SnnTorch, et évaluées à l'aide de deux ensembles de données différents. L'évaluation des trois approches a été effectuée sur deux ensembles de données différents, en utilisant six fonctions de perte différentes, et en utilisant la précision comme mesure principale. Ainsi, de nouveaux horizons d'applications sont ouverts, ainsi que de nombreuses techniques d'optimisation possibles.

Mots clés : Réseau de neurones à impulsions, Rétropropagation du gradient, Descente de Gradient, Algorithmes neuromorphiques

---

Abstract - Spiking Neural Networks offer a new approach to tackle the efficiency issues of conventional artificial intelligence (AI), but there is a big performance gap between Spiking Neural Networks and Artificial Neural Networks. This work contributes to the optimization of Spiking Neural Networks in order to reduce this gap. The proposed methods are focused on architecture optimization and the development of a new spike-based loss functions, to prevent the dead neuron issue. The proposed techniques are implemented using PyTorch and SnnTorch frameworks, and benchmarked using two different data sets. The evaluation of the three approaches has been done on two different data sets, using six different loss functions, and using accuracy as a main metric. Hence, new horizons of applications are opened up, as well as many possible optimization techniques.

Index-terms : Spiking neural networks, Backpropagation, Gradient descent, Neuromorphic computing

---

CONTENTS

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence. 14, 17, 56, 86

ANNs Artificial Neural Networks. 13, 14, 17, 19, 31, 34, 36–38, 42–46, 49, 50, 62, 71, 84, 86, 88

BNTT Batch Normalization Through Time. 43

BP Backpropagation. 17, 30, 31, 34–36, 50

BPTT BackPropagation Through Time. 34–36, 45, 88

CE Cross Entroppy. 39, 42, 53, 67

CNNs Convolutional Neural Networks. 83, 86

DVS Dynamic Vision Sensor. 58

IF Integrate and Fire neuron. 24, 31

LIF Leaky Integrate and Fire neuron. 25, 28, 30, 46

ML Machine Learning. 13, 14, 17, 86

MSE Mean Squared Error. 39, 41, 42, 53

nLIF nonLeaky Integrate and Fire neuron. 30

ReLU Rectified Linear Unit. 31, 38

RNNs Recurrent Neural Networks. 30, 34, 43, 46, 83, 84, 88

SNNs Spiking Neural Networks. 14, 17–19, 23, 30–32, 34–36, 38–40, 42–47, 49, 50, 56, 57, 62, 64, 67, 68, 71

STDP Space Time Dependent Plasticity. 31–33, 36

tdBN threshold-dependent Batch Normalization. 43

INTRODUCTION

Motivations

The overwhelming success of AlexNet in 2012 is a historical landmark in AI history [61], and the start of a new era, which is Machine Learning (ML) and ANNs. ANNs became remarkably successful in solving challenging problems and outperformed humans in a variety of tasks that depend upon high-dimensional data. This achievement is typically attributed to the gradient descent algorithm and backpropagation, and strongly relies on the availability of large amounts of data and massive computing resources [33].

However, the amount of energy and computational power required to train a state-of-the-art algorithm continues to grow at an exponential rate (10×per year between 2012 and 2019) [2, 36] and becomes a real constraint in recent times; this fact is well illustrated in the history of ML algorithms in the last decade.

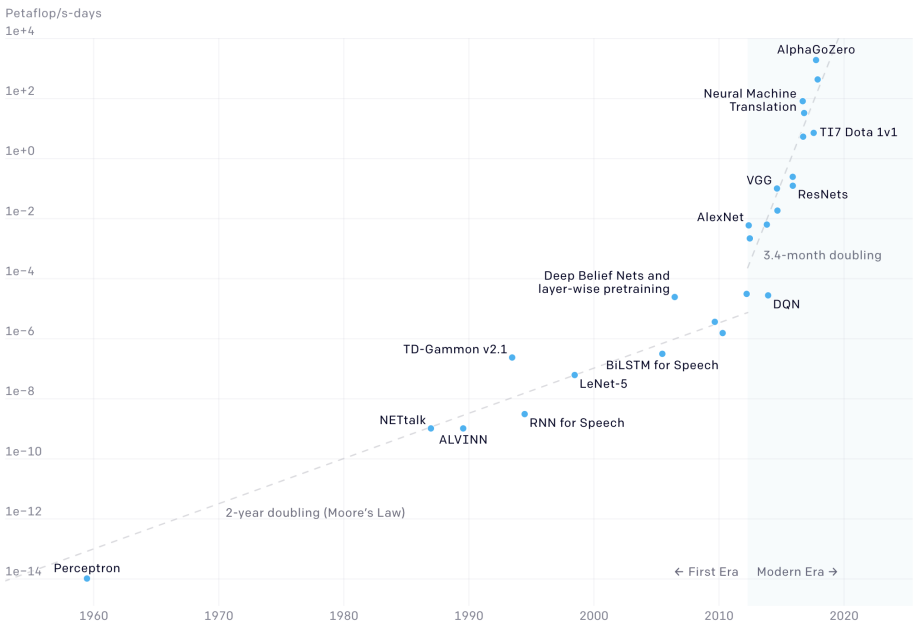

Figure 1: AI computing resource and energy requirement over decades [2].

At a given time, the development and implementation of a more efficient algorithm takes place after a few months or weeks. But over time, this has become very difficult to achieve because of the energy consumption required, which implies a significant development cost. A common example is GPT-3 a language model developed by OpenAI, among the most renowned deep learning algorithms in the last three years, contains about 175 billion parameters, costs approximately 12 million US dollars to train [62], with an estimated 190,000 kWh overhead in energy consumption [4, 13]. In contrast, the brain consumes only about 12-20W, unequally considered as the best reference compared to all state-of-the-art models.

Cost, energy consumption, as well as the substantial need for usage ANNs on resource-limited devices such as phones, is driving researchers to find new avenues to optimize ANNs and ML algorithms. Among the proposed approaches are SNNs, bio-inspired ultra-low power algorithms, commonly referred as the third generation of Artificial Intelligence (AI) [65].

These algorithms constitute a potential solution, as they are more biologically realistic than ANNs, more hardware friendly and energy efficient, and thus are appealing for use on tiny and portable devices. Yet, they still trail behind ANNs in terms of performance.

Aims and Objectives

This dissertation investigates how to bridge this performance gap between SNNs and ANNs for classification problems. The first goal of this work is to explore and understand the concepts of SNNs, as well as previous research conducted in this field. This thesis goes through the details of these models by explaining the theory behind their concepts and giving clear steps for their architecture, then identifying their weaknesses to optimize them. However, the main objective of this work is to innovate new approaches for information processing and model training in order to optimize the final accuracy performance of these models.

1. Propose new approaches for information processing to optimize accuracy and loss.
2. Evaluate the proposed techniques using different datasets.
3. Compare the resulting performance with the baseline models.

Description of the work

This thesis is structured as follows :

- **Theoretical part :**
  - **Chapter 1 :** This chapter goes into the theory behind Spiking Neural Networks, then a comparison between SNNs and ANNs in order to highlight the difference and similarities between them. Then it highlights the drawbacks of SNNs that cause the performance gap.

- **Chapter 2 :** This chapter goes into the details of the proposed methods, the negative threshold, the adaptive threshold, and the new loss function, explains the principle and how they can contribute to the optimization of the model, and discusses related work.

- **Experimental part :**

  - **Chapter 3 :** This chapter is dedicated to the realization and evaluation of the optimized models in this thesis. First, we present the tools, frameworks, and databases used in this work. Finally, we present the evaluation protocol that has been used to capitalize on the importance of our methods.
  - **Chapter 4 :** This chapter presents the experimental results of the baseline models and the proposed models using six different loss functions and two different data sets. Then, we discuss the present results and analyze them. Finally, we select the best models on each data set and compare their performance with previous work.

- Finally, we have concluded this thesis with a statement of the challenges encountered during the development, while providing possible solutions to improve the results obtained and new perspectives to be exploited in future work.

Part I:

Theoretical part

> The question is whether you are better off exploiting the progress of digital technology or try to follow the neuromorphic philosophy. I am interested, but I am sceptical.
>
> Yann LeCun

CHAPTER 1

SPIKING NEURAL NETWORKS

The aim of this chapter is to familiarize the reader with the theoretical concepts that are behind the development of spiking neural networks. This part assumes that the reader has basic knowledge of AI and ML and understands the Backpropagation (BP) algorithm (Appendix A 4.5). First, it recalls some basic differences between ANNs and SNNs, including information coding and processing using a different activation function. Then, we investigate the possible options to train a SNNs using different learning methods. Then, we discusse the evaluation metrics that are used to compare with the base-line models. Finally, we conclude with the unsolved problems that cause the gab between ANNs and SNNs.

1.1) SNNs vs ANNs

Although ANNs are historically brain-inspired, they are still very far from properly representing the neural system. This is why SNNs is getting more and more attention; Spiking Neural Networks are just more adapted versions of the above. They share similar topology, as well as most of the mathematical theory [66, 61]. However, the fundamental difference is about coding and processing of information within the network [20]. The figure fig. 1.1 is an example of SNNs networks and its differences from ANNs.

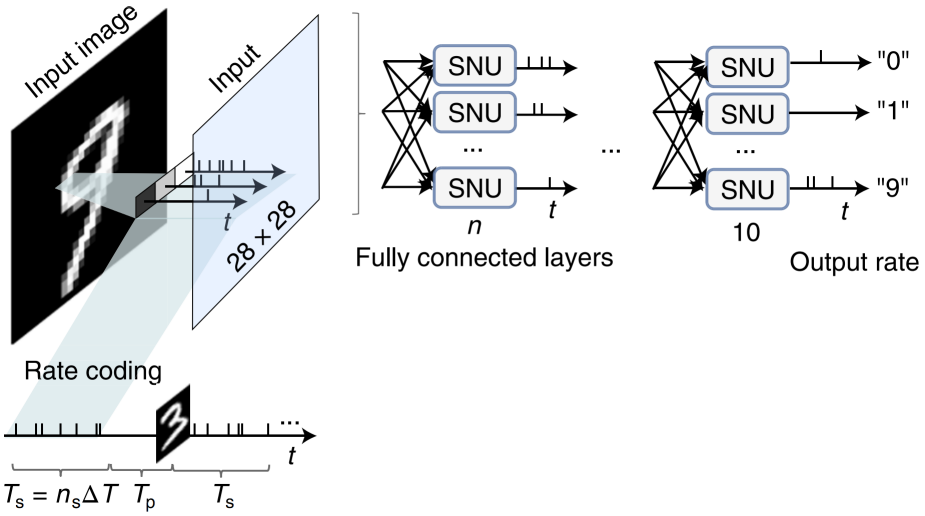

Figure 1.1: Fully connected spiking networks processing a static input image during T_s [66].

1.1.1) Neural coding

The neural code refers to how the brain represents information [61]. Although there are many theories, the code has not yet been cracked [20]. Experimental brain observations have shown that information is shared as a train of action potential through synapses [64]. An action potential is approximated as a uniform spike with $100mV$ of amplitude and $1msec$ of width [64, 61]. Therefore, SNNs aim to code the information using the notion of action potential.

However, SNNs are event-driven networks, so the occurrence of an action potential is much more important than the subtle variations of the action potential [61]. As a result, most models represent a spike as a discrete single bit, mathematically equivalent to a Dirac pulse. An all-or-nothing event significantly reducing the calculation due to multiplication [61]. For example, instead of calculating the weighted action potential, which requires a lot of computation and leads to a carry propagation delay, we have just multiply the weights by a single bit. This trades off the cumbersome multiplication process to a simple read-out of the weight value in memory [61]. The figure fig. 1.2 illustrates a spiking neuron receiving encoded information and processing it.

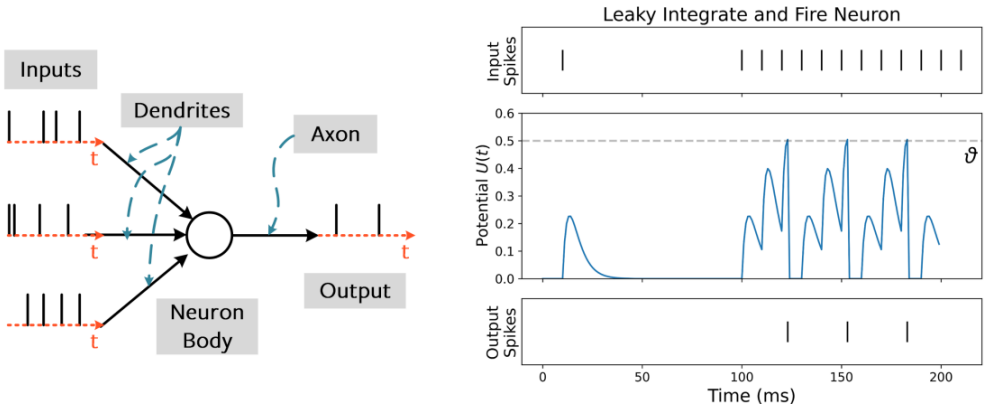

Figure 1.2: Illustration of a spiking neuron, where input spikes are passed to the neuron and sufficient excitation pushes the neuron to spike on his turn [20].

The usage of impulses implies adding the time dimension into consideration, which allows various encoding modes, but regardless of how the information is conveyed by a spiking event, the use of binary pulses allows adding more sparse in time [53]. Here lies the main difference, where ANNs considers the output of the neuron constant until the next pass, so the time dimension is usually ignored. Yet, for Spiking Neural Networks, the information depends on time. Thus, it implies three major advantages of SNNs (with respect to ANNs):

- Information is encoded in both space and time, which encourages the detection of a spatiotemporal pattern [58].
- From the perspective of SNNs, ANNs is continuously spiking, and spike events consume energy. However, SNNs uses only a few spike events and reduces energy consumption [58, 53].
- Spikes are high-density information events. Therefore, they can multiplex information into a single stream of signals, such as the frequency and amplitude of sound in the auditory system [64].

Here are the commonly used coding schemes in the figure fig. 1.3 (this is not an exhaustive list):

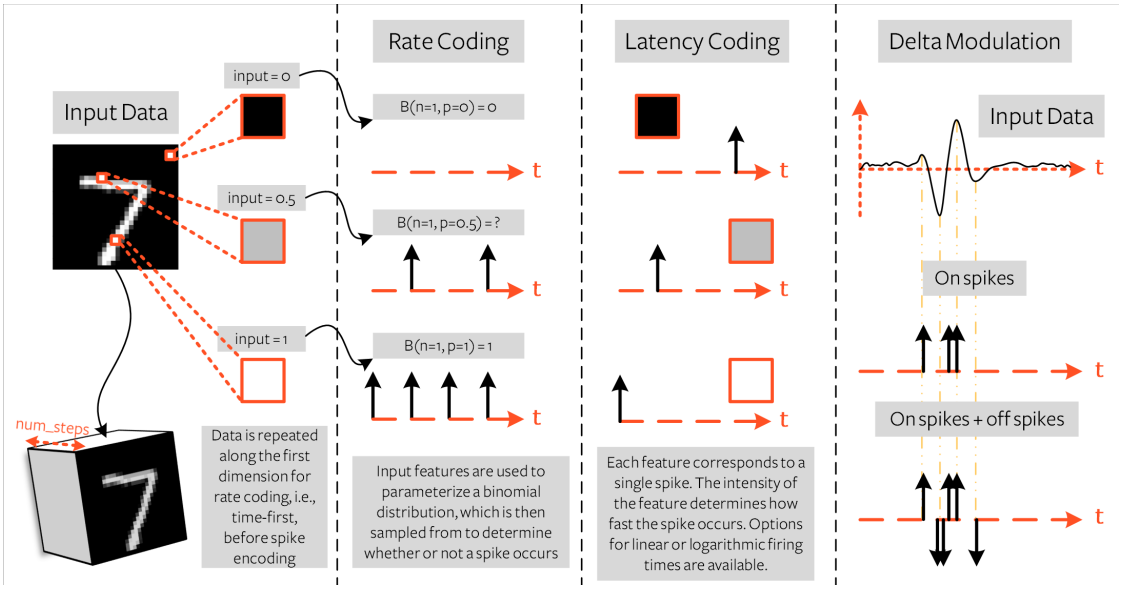

Figure 1.3: Illustration of the possible coding schemes [20].

1.1.1.1) *Rate coding*

A bioinspired mechanism that conveys the information value into rate or frequency over a time interval, and the number of spikes produced is correlated with the value [24, 25].

From spikes to a value: Used for converting the output layer results into interpreted values. It can be further divided into two subcategories:

- **Count rate:** This is the most forward approach; it can be done by counting all spikes produced by a single static input or by averaging the number of spikes by the time window of a single dynamical stimulus [20].

$$\mu = N_{spikes}(T) \qquad \mu = \frac{N_{spikes}(T)}{T} \tag{1.1}$$

  Here T is the number of time steps, and N_{spikes} is the sum of spikes.

- **Density rate:** An alternative approach [5, 20] is to constrain the counting window to an infinitesimally short duration Δt, such that only a single spike may possibly occur within the window for a given neuron, which constrains the output to be either '1' or '0'. However, the process is repeated K times, until the average is equal or close to 1 (interpreted as the probability of neuronal firing within an infinitesimally small period of time). The value is the total sum of spikes N_{spikes}, is weighed by the infinitesimal time window duration, then divided by the number of iterations.

$$\mu = \frac{1}{\Delta t} \frac{\sum_k^K N_{spikes}(T)}{K} \tag{1.2}$$

From a value to spikes: This coding is used to convert the input value to spiking values that will be applied on the input layer. There is no exact formula; spikes are generated using a probability distribution with a spiking probability given by the normalized pixel intensities. In practice, converting a static image into a dynamical one is done by taking several Bernoulli samples of the image to create certain dynamics. Mathematically, above a certain percentage, the distribution allows us to create a representative sample of our data; thus creating certain dynamics without repeating the same image each time. This process is known as Poisson encoding [33].

1.1.1.2) *Latency coding*

Many researchers noticed that a human visual system needs about $150ms$ to process object recognition tasks, and such a speed response cannot be supported by a rate code [65, 20], similar conclusions are made on other olfactory systems. Suggestions are toward embedding information in the firing time, also known as temporal coding or latency coding [10], where the highest value leads to the lowest spike latency and vice versa.

Conversion by convention: The conversion formula between the firing time and a value, either in the input or in the output, can be fixed by convention as [20]:

- Non-linear formula:

$$t_i = \frac{N}{v_i} \tag{1.3}$$

- Linear formula:

$$t_i = 1 - \frac{v_i}{N} \tag{1.4}$$

Where v_i is the value normalized by the highest value in an image N.

Bio-inspired conversion: The function can also be biologically inspired and depends directly on the neuronal model. Considering the Leaky Integrate Fire model that admits the following formula (Demonstration in the next section 1.1.2.1):

$$U(t) = I_{in}R + (U_0 - I_{in}(t)R)\, e^{-\frac{t}{\tau}} \tag{1.5}$$

As result:

$$U(t) = V_{thre} \qquad \Longrightarrow \qquad t = \tau ln\left(\frac{I_{in}R}{I_{in}R - V_{thre}}\right) \tag{1.6}$$

This coding, can be very beneficial in the case of widely stimuli. Logarithmic function helps in compressing information.

An important advantage of this coding is that it codes all the information in a single spike, which encourages more sparsity in the network. Furthermore, the mechanism seems more biologically plausible. It is important to note that there are other latency coding schemes, such as considering the relative time between spikes.

Comparison between Rate and Latency coding

	RATE CODING	**LATENCY CODING**
ADVANTAGES	• Error tolerance, if a neuron file to fire, there are other spikes to reduce the error effect. • Encourage learning, since more spiking means less occurrence of dead neurons, which means better learning.	• Very quick convergence and learning, due to sparsity there is less memory access. • Also, due to sparsity, there is less energy consumption.
DRAWBACKS	• Higher energy consumption (inefficient compared to latency coding). • Require several time steps or high rate to average out discretization noise.	• Risk of dead neurons, since the difference between sparsity and dead neurons is very fine.

Table 1.2: Table of comparison between rate and latency coding schemes.

1.1.1.3) *Population coding*

It is a particular type of coding that is usually combined with latency or rate coding to combat dead neurons and the vanishing gradient (description of dead neurons and the vanishing gradient in section 1.2.5). The principle is that a class is represented by several neurons in the output layer, and the value is the average of the coding result on the number of neurons in the class [12]. This encourages Backpropagation (BP) to have different paths to update the network in case of the occurrence of a dead neuron in the output layer [20]. It also helps reducing the discretization noise in rate coding [49].

1.1.1.4) *Delta modulation*

Also known as Send-on-Delta encoding, it is a spike generation mechanism inspired from neuroscience theory [47], suggesting that the human brain undergoes a sensory suppression, which states that the brain only focus on variation of an object when faced with multiple objects simultaneously. Mathematically, this coding is event-driven sampling where thresholding is applied in the signal acquisition. To thrive on change, it captures only the difference between inputs, not frames, and the difference is considered only if it is beyond a pre-defined threshold.

A spike is generated at time t_k only if the difference between the actual value and the value of the previous spike time t_{k-1} is greater than the threshold Δ [76] which is the sensitivity.

$$t_k = \{t, \quad \Delta \leq |x(t) - x(t_{k-1})|\} \tag{1.7}$$

$$\Delta = w^2 \tag{1.8}$$

Variations are coded into channels, one for increasing luminance (on-spike) where $0 < w$ is positive and a second channel for decreasing luminance (off-spike) where $w < 0$ is negative. The figure fig. 1.4 illustrate coding information variation into two different channels, and the figure fig. 1.5 illustrate the differences between conventional camera and neuromorphic one.

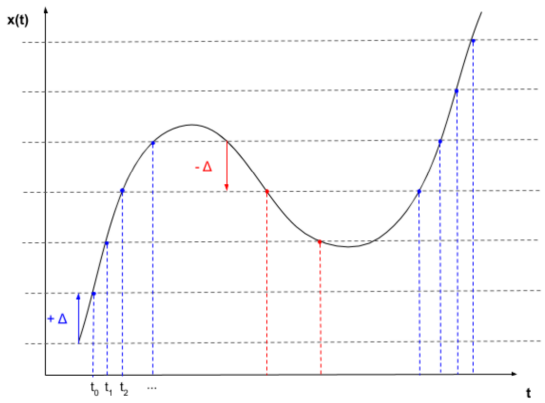

Figure 1.4: Send-On-Delta sampling strategy. Blue dots represent sampling due to a significant increase, red dots represent sampling due to a significant decrease [76].

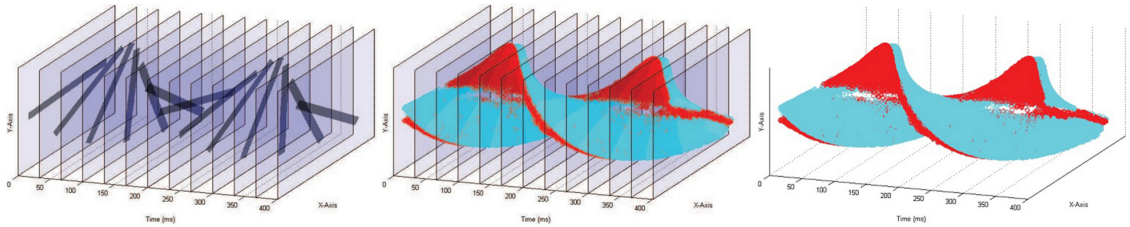

Figure 1.5: Right: Recording of a rotating bar captured with a neuromorphic camera. Blue and red points indicate on (increasing intensity) and off (decreasing intensity) events. **Left:** Conventional camera recording, which is frame based. **Middle:** Superposition of the NV and CV data [60].

The variation that a neuromorphic camera captures with respect to time is very sparse and difficult to interpret, even by humans. Generally, an integration of time intervals is applied to produce frames. Those frames are equivalent to thresholding on the successive frame difference of conventional cameras. The figure fig. 1.6 demonstrate the difference between conventional camera recording and neuromorphic camera recording.

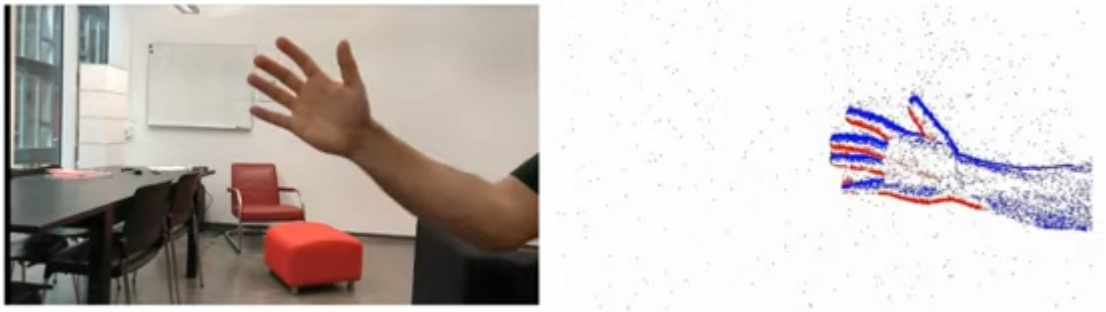

Figure 1.6: Example of hand moving frame captured by a neuromorphic camera using Delta modulation [23].

NOTES

- Although dynamical coding of information is beneficial and preferred for a spiking network, it is also possible to use static inputs directly by repeating each input for a certain pre-fixed number of time steps. However, the input in this case is considered as a direct current input over time, and fails to exploit the SNNs properties at the input layer.
- Rate and Latency coding are used to convert static datasets into dynamical ones, but real neuromorphic datasets are directly encoded by a delta modulation at the acquisition phase.
- Why convert static images? The first goal is to recognize the event streams coming from a real silicon retina sensor (neuromorphic camera). Due to the lack of a convenient benchmarking dataset, traditional image datasets with rate/latency conversion were used in the early stages of SNNs development [58]. Although a real neuromorphic dataset is not a spike train but contains dynamic spatio-temporal patterns of events, it helps to simulate and train the network.

1.1.2) Neural processing

Much like its counterparts, the spiking neuron processes a weighted sum of inputs. However, the input is different, so the processing mechanism differs as well. These processing units modulate information by mimicking a complex biological process that relies on a multitude of ionic currents inside and outside the cell membrane [30].

The figure fig. 1.7 illustrate types of neuronal model. Hodgkin and Huxley [29] propose the first detailed model; It is a very accurate biophysical model, but it is complicated and requires a high computational cost. However, several suggestions are proposed to overcome this limitation, phenomenon-oriented models that exclude unnecessary details, also known as phenomenological models, which are an accurate tradeoff between reduced computational cost and biological realism.

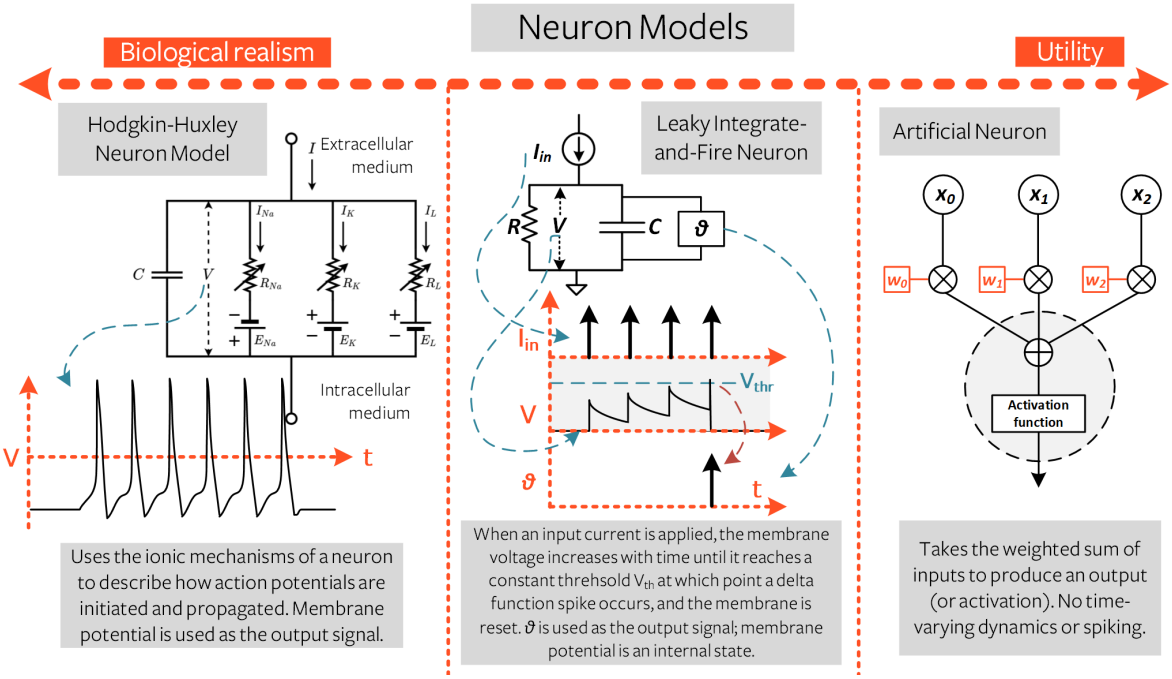

Figure 1.7: Illustration of different neuronal models Hodgkin and Huxley, SNNs, ANNs [20].

Integrate and Fire neuron (IF) models are inspired by the research of Louis Lapicque in 1907 on neural dynamics [40, 20], he has concluded that a spiking neuron resembles coarsely a low-pass filter circuit. However, depending on the filter and the approximation approach, several varieties exist, the most important ones being the following.

1.1.2.1) *Leaky Integrate and fire neuron*

The neuron is approximated as a low-pass filter circuit consisting of a resistor R and a capacitor C. The dynamics does not take into account the reset mechanism. The dynamics are modeled with the following circuit illustated in the figure fig. 1.8.

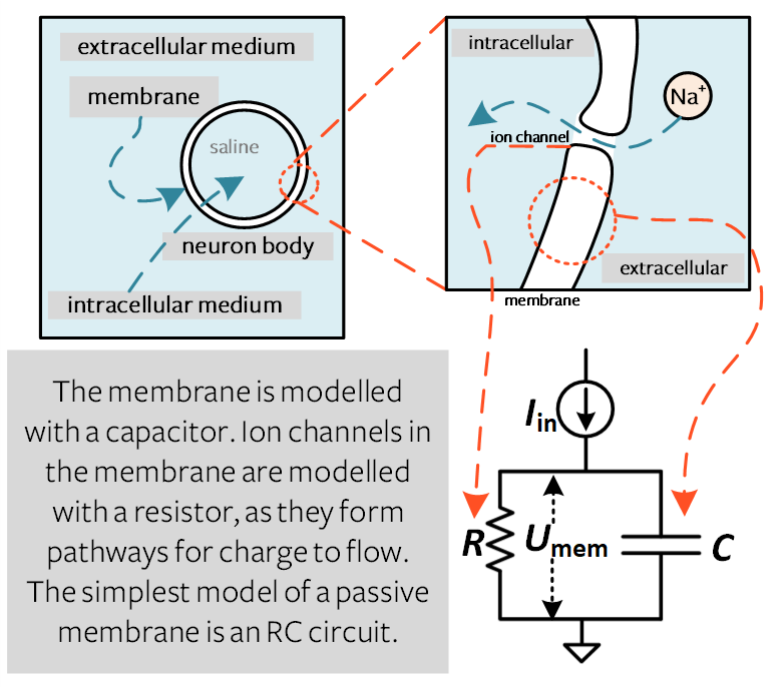

Figure 1.8: RC filter model of LIF neuron [20].

The circuit can be represented with equation eq. (1.9):

$$\tau \frac{dU(t)}{dt} = -(U(t) - U_{res}) + I_{in}(t)R \tag{1.9}$$

Where:

U: The capacitor potential.

U_{res}: The resting potential is assumed to be 0 for simplicity.

I_{in}: The input current.

$\tau = RC$: The time constant, a propriety of the circuit, is on the order of 1 to 100 milliseconds.

It is a homogeneous differential equation and can be solved using the variation of parameters method:

- **General solution:** Used to find a general form of the solution:

$$\tau \frac{dU(t)}{dt} = -U(t) \tag{1.10}$$

  So the solution has the form:

$$U(t) = U_0 e^{-\frac{t}{\tau}} \tag{1.11}$$

- **Particular solution:** Considering injecting this general solution into equation eq. (1.9):

$$U(t) = K(t) e^{-\frac{t}{\tau}} \tag{1.12}$$

$$\frac{dU(t)}{dt} = K'(t)e^{-\frac{t}{\tau}} - \frac{K(t)}{\tau}e^{-\frac{t}{\tau}} \tag{1.13}$$

$$= (K'(t) - \frac{K(t)}{\tau})e^{-\frac{t}{\tau}} \tag{1.14}$$

$$\frac{dU(t)}{dt} = -U(t) + I_{in}(t)R \tag{1.15}$$

$$\tau\left(K'(t) - \frac{K(t)}{\tau}\right)e^{-\frac{t}{\tau}} = -K(t)e^{-\frac{t}{\tau}} + RI(t) \tag{1.16}$$

$$\tau K'(t)e^{-\frac{t}{\tau}} = RI(t) \tag{1.17}$$

$$K'(t) = \frac{R}{\tau}I(t)e^{\frac{t}{\tau}} \tag{1.18}$$

$$K(t) = \frac{R}{\tau}\int_0^t I(t)e^{\frac{t}{\tau}}dt \tag{1.19}$$

The exact solution depends on the expression of $I(t)$.

However, since in practice the hardware is numeric, the time is sequence-based and discrete, the Euler forward method can be used to find an approximation for small values of Δt:

$$\tau\frac{U(t+t) - U(t)}{\Delta t} = -U(t) + I_{in}(t)R \tag{1.20}$$

$$U(t+t) = \left(1 - \frac{\Delta t}{\tau}\right)U(t) + \frac{\Delta t}{\tau}I_{in}(t)R \tag{1.21}$$

Where:

Δt: The discrete time difference between successive sequences.

$\beta = 1 - \frac{\Delta t}{\tau}$: The leak effect between time steps, also known as the decay rate.

Thus:

$$U(t) = \beta U(t-1) + (1-\beta)RI_{in}(t) \tag{1.22}$$

The input current is equivalent to the sum of weighted spikes input, but the weight values can absorb the effect of $(1-\beta)$ by scaling, the same applies to the resistance value R. Then we have writed:

$$U(t) = \beta U(t-1) + WX(t) \tag{1.23}$$

NOTES

- The expression of the decay is appropriate only if $\Delta t << 1$.
- Since the decay rate is a propriety of the circuit and independent of the input value, assuming that the spiking neuron does not receive any input $X = 0$, in this case the membrane potential is a pure exponential decay, subsequently:

$$U(t) = U_0 e^{-\frac{t}{\tau}} \tag{1.24}$$

Since it concerns a discrete time, then:

$$U(t = n\Delta t) = U_0 e^{-\frac{n\Delta t}{\tau}} \qquad \Longrightarrow \qquad \beta = e^{-\frac{\Delta t}{\tau}} \tag{1.25}$$

This expression is more useful since it works regardless of the value of Δt

Until now, the circuit model does not include a reset mechanism. The reset is modeled by a non-physical component that have the effect of an "IF" statement. In equation, it can be done by two ways illustrated in the figure fig. 1.9.

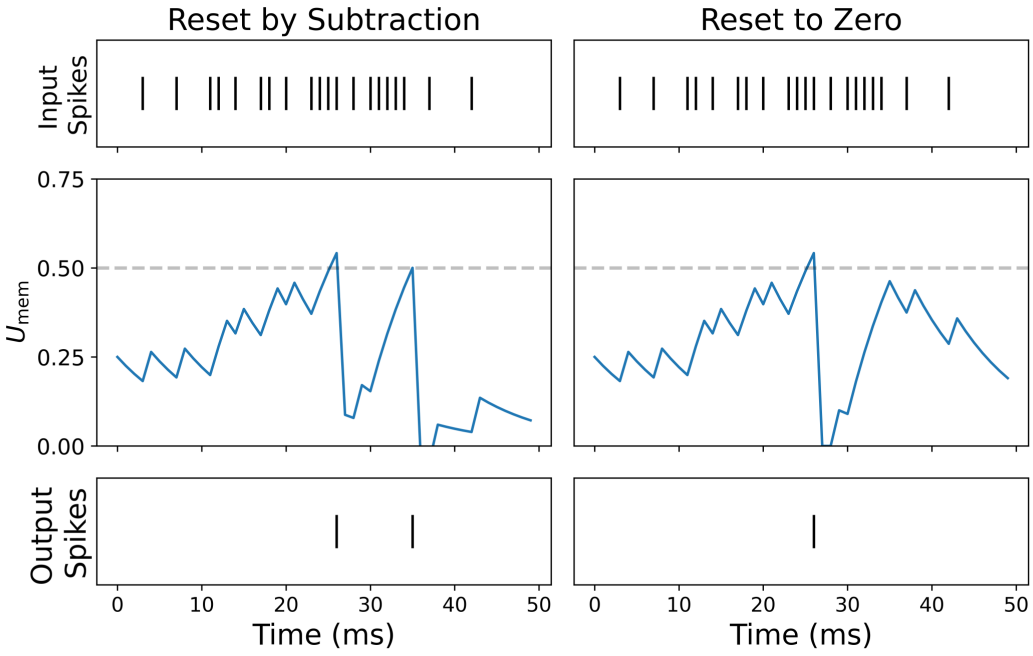

Figure 1.9: Comparison between rest by subtraction and reset to zero [20].

- **Reset to zero:** By forcing the membrane potential to zero at the onset of a spike.

$$U(t) = \begin{cases} \beta U(t-1) + WX(t) & \text{if } S_{out} = 0 \\ 0 & \text{Else.} \end{cases} \tag{1.26}$$

$$S_{out}(t) = H(U(t) - \theta) \tag{1.27}$$

- **Reset by subtraction:** By deleting the threshold value at the onset of a spike.

$$U(t) = \beta U(t-1) + WX(t) - S_{out}(t-1) \times \theta \tag{1.28}$$

$$S_{out}(t) = H(U(t) - \theta) \tag{1.29}$$

Experimentally reset by subtraction has better performances [20], and this is why it is used in the rest of this work.

By analogy:

U: The membrane potential is equivalent to the capacitor potential.

β: The decay rate, its value between $0 < \beta < 1$.

W: The weights control the impact of the spikes, equivalent to the impact of the resistor on the potential.

X: The input spikes are equivalent to the input current.

H: The Heaviside step function.

S_{out}: The output spike.

θ: The threshold for the reset mechanism in the event of a spike.

Overall, the behavior is summarized in the unrolled computational in the figure fig. 1.10, and follows those stages [69]:

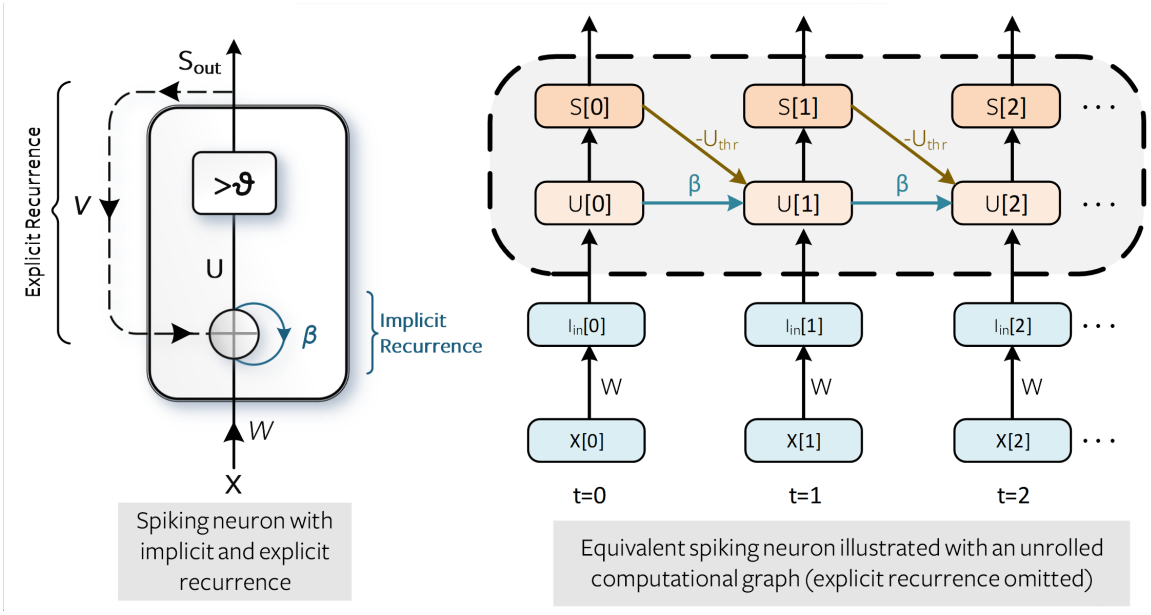

Figure 1.10: Computational graph of LIF neuron [20].

- At the initial stage, in the absence of input excitation $X = 0$, the neuron is inactive, the membrane potential U is null, and so is the output.

- When inputs are applied, the neuron adds the weighted sum of those inputs to the membrane potential.
- However, the inputs are considered as a quick short excitation pulse that charges the capacitor and increases its potential. The capacitor potential is the membrane potential [64] and due to the behavior and leaks of the capacitor, it is not constant, but it decays over time in exponential relaxation. The leak is determined by the decay rate β, which controls the intensity of relaxation.
- If the membrane potential crosses above a certain threshold, a spike is emitted as a spike to the subsequent connections, and the membrane potential is reset.

1.1.2.2) *non-Leaky Integrate and Fire neuron*

A more drastic approximation suggests that a neuron can be modeled as a perfect capacitor C [31] with a threshold mechanism. It can be approximated as follows:

$$C\frac{dU(t)}{dt} = I_{in} \tag{1.30}$$

In practice the hardware is numeric, the time is sequence-based and discrete, the Euler forward method can be used to find an approximation for small values of Δt:

$$\frac{U(t+\Delta t)-U(t)}{\Delta t} = \frac{I_{in}(t)}{C} \tag{1.31}$$

$$U(t+\Delta t) = U(t) + \frac{\Delta t}{C} I_{in}(t) \tag{1.32}$$

Where:

U: The capacitor potential.

I_{in}: The input current.

C: The capacitance.

The capacitor can be ignored since inputs are weighted, and it is possible to assume that weights delete the capacitor scaling effect. Also, the modeling has so far not included the reset mechanism. By adding the rest mechanism:

$$U(t) = U(t-1) + WX(t) - S_{out}(t-1) \times \theta \tag{1.33}$$

$$S_{out}(t) = H(U(t)-\theta) \tag{1.34}$$

Where:

U: The membrane potential is equivalent to the capacitor potential.

W: The weights control the impact of the spikes, equivalent to the impact of the resistor on the potential.

H: The Heaviside step function.

X: The input spikes are equivalent to the input current.

S_{out}: The output spike.

θ: The threshold for the reset mechanism in the event of a spike.

To recapitulate this dynamics, just as LIF the nonLeaky Integrate and Fire neuron (nLIF) accumulates the weighted sum of the inputs, it contributes to the membrane potential. Although the membrane potential here does not present any leaks $\beta = 1$, so its value is never forgotten until it reaches the threshold. In this case, the membrane potential is a pair of constants with discontinuities, making the analytical calculations for BP more difficult [14].

NOTE

From the equation of SNNs, it is obvious that it behaves as RNNs where the state and the output of the neuron depend on its previous state or the state of neurons in the same layer or after it, i.e., cross recurrence among neurons of interlayer and interneuron connections. As a result, SNNs are a particular case of RNNs (illustrated in the figure fig. 1.11) where it has a self-recurrence connection within each neuron, while RNNs have cross-recurrence among neurons.

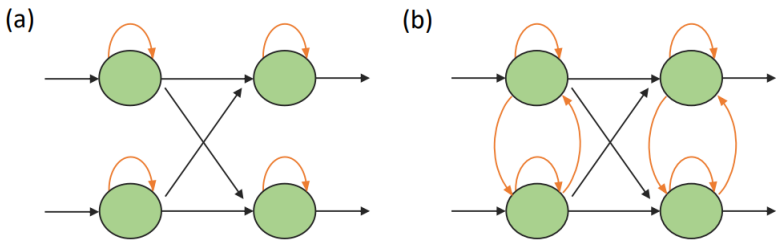

Figure 1.11: Connection pattern of (a) SNNs and (b) RNNs [28].

1.2) Training SNN

Multiple training mechanisms are available, from supervised to unsupervised. They are mathematically based methods like BP or biologically motivated like Spike Timing Dependent Plasticity (STDP), as well as many hybrid approaches.

1.2.1) Shadow training

In the lack of an accurate way to use BP for SNNs during the last decades, conversion techniques gained a lot of attention by adapting a pre-trained ANNs parameters to fit into SNNs. This approach is driven by the observation that a IF spiking neuron is functionally equivalent to a Rectified Linear Unit. The relationship can be expressed as follows [20]:

$$S_{out} = H(U-\theta) = \frac{\text{ReLU}(U-\theta, 0)}{U-\theta} \tag{1.35}$$

Where:

U: The membrane potential.

H: The Heaviside step function.

S_{out}: The output spike.

θ: The threshold.

Using this approach, several models have been developed that have achieved competitive accuracy performance, but the conversion loss is still considerably high. For instance, consider an example of conversion of a Rectified Linear Unit (ReLU) function that receives two inputs of value 1 and 0.5, their sum is 1.5. Meanwhile, the threshold of spiking neuron should be fixed at the highest value, otherwise the spiking neuron keeps firing. In this case, the threshold is fixed at 1.5, but the input conversion to spike using a rate coding implies that the first input fire frequently assuming 100% and the second would fire only half of the time 50% in the time window. The addition of both would give 1 or 2 at the membrane potential. In both cases, there is an error of ± 0.5. Some information loss would take place and impact the final loss of the network.

In the case of rate coding, several approaches are suggested to overcome this issue, threshold balancing [38], weight normalization [53], binarization [45]. It is worth mentioning that temporal dynamics are rarely considered in the process of training for converted SNNs [38], except for this work [73] where they encode the ReLU output as latency coding directly.

NOTE

The choice of threshold value can be ambiguous, spike time is proportional to the threshold value; this way the network takes longer delay without spiking, in deep architecture these delays accumulate into long periods, and cause a significant dead neuron's problem (description in section 1.2.5). Meanwhile, a lower threshold reduces the ability of SNNs to distinguish between different magnitudes of spike input that accumulate in the membrane potential, causing an accuracy degradation. A careful and strategic choice of the threshold is essential [53].

Regardless of advantages and drawbacks, this technique was the only approximation to validate the concept of spiking neural networks until recently, shadow trained SNNs is very unlikely to reach the performance of the original network, but this is not an issue since there are other learning methods.

1.2.2) Spike Timing Dependent Plasticity

Synaptic connections are the information support and play a major role in it preprocessing, whether excitatory or inhibitory way. Synaptic plasticity is also the preprocessing change usually refers to aptitude of learning.

Spike Timing Dependent Plasticity (STDP) is a biologically motivated learning rule that stems from the experimental results of [57, 46], where they measured the postsynaptic potential (initial slope) of different neurons under stimulation, then they plotted the variation of the synaptic weights with respect to relative timing between presynaptic spike arrival and postsynaptic firing. The pairing is repeated about 100 times with a frequency of 10Hz. The figure fig. 1.12 is the resulted plot from the previous experiment.

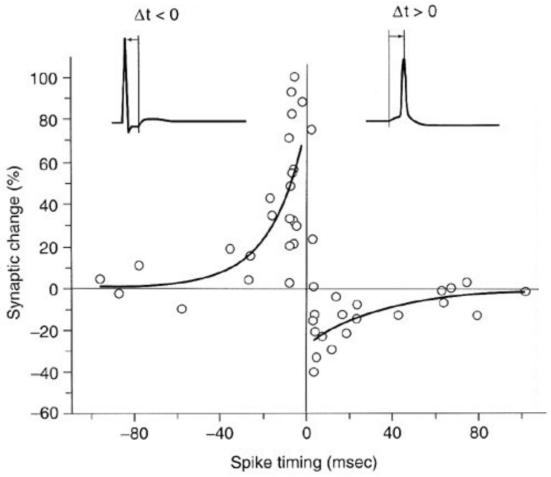

Figure 1.12: STDP learning curve.

The plot produced formerly known as the STDP learning window uses relative spike timing between neurons for unsupervised training of SNNs, where weight updates are proportional to causality relationship between the pre- and postsynaptic spikes. Thus, STDP strengthen the weights when two events occur in the expected causal temporal order and weaken them

otherwise by modulating it with an exponential decay [a] [64]. Due to this mechanism, neurons are highly sensitive to spike times, leading to competition among presynaptic neurons, resulting in shorter latencies and spike synchronization [1, 59]. The figure fig. 1.13 indicates the approximated learning curve of STDP.

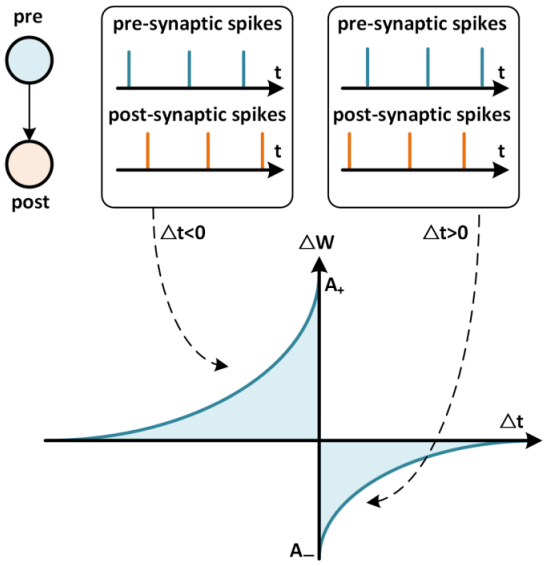

Figure 1.13: STDP learning curve [20].

Mathematically, the result can be written as:

$$\Delta w = \begin{cases} A_+ e^{-\frac{\Delta t}{\tau_+}} & \text{si } \Delta t < 0 \\ A_- e^{\frac{\Delta t}{\tau_-}} & \text{sinon } 0 < \Delta t \end{cases} \tag{1.36}$$

Where:

$$\Delta t = t_{post} - t_{pre} \tag{1.37}$$

NOTES

- This learning rule correlates with the Hebb postulate "Neurons that fire together, wire together".
- Different types of synapses can have different forms of STDP learning window [1, 8].
- Although STDP is the basis of many unsupervised learning approaches [18]. It is worth mentioning that formal STDP is rarely used for training [61]. For simplicity or to satisfy convenient mathematical properties, there are other variations of STDP, especially those combined with gradient-based methods, where STDP is used in the backward pass to modulate the loss of the forward pass [48].

[a] "Strengthening is called long-term potentiation (LTP) and weakening is called long-term depression (LTD). The phrase long term is used to distinguish between very transient effects on the scale of a few ms that are observed in experiments" [61].

1.2.3) Backpropagation and Gradient Descent algorithms

Likewise ANNs, Backpropagation (BP) algorithm is the current state-of-the-art method for supervised training of SNNs. It carries out credit assignment of the output loss to each parameter of the network by implementing the chain rule of differentiation. Credit assignment can be either spatial in which the update is assigned spatially to targeted parameters, or spatiotemporal that target the parameter at particular temporal duration. For a long time, BP could not be applied due to the non-differential nature of the spiking neuron, different researches have been carried out to bypass this challenge.

1.2.3.1) *Backpropagation using spike time*

First introduced by [11] as SpikeProp, to overcome BP issues due to thresholding discontinuities, the nondifferentiable was traded by introducing the spiking time to the chain rule, where spikes can be discontinuous, but time is continuous [20]. Subsequently, the network uses a latency coding where a neuron spikes only once.

$$\frac{d\mathfrak{L}}{dW} = \frac{d\mathfrak{L}}{dt_i}\frac{dt_i}{dW} = \frac{d\mathfrak{L}}{dt_i}\frac{dt}{dU}\bigg|_{t=t_i}\frac{dU}{dW}\bigg|_{t=t_i} \tag{1.38}$$

Where for a neuron j receiving an input of value X_i, which is the sum of all spiking neurons of the presynaptic neuron i modulated by weight $W_{i,j}$:

$$U = \sum_i W_{i,j} X_i \tag{1.39}$$

$$X_i = \sum_k X_i^k = \sum_k \epsilon(t - t_i^k) \tag{1.40}$$

As result:

$$\frac{dU}{dW} = X_i = \sum_k \epsilon(t - t_i^k) \tag{1.41}$$

Assuming a linear approximation of the activation function around $t = t_i$, where $dU = \alpha dt$, we have the following:

$$\frac{dt_i}{dU} = \frac{1}{\alpha}\left(\frac{dU}{dt_i}\right)^{-1} \tag{1.42}$$

The final equation that links all depends on the loss function used and the spiking neuron model.

1.2.3.2) *Backpropagation Through Time*

Alternatively SNNs can be perceived as subclass of RNNs, since the equation of the spiking neuron shows an important time dependence, where the membrane potential constitutes the hidden state. In this case, SNNs can be trained by BackPropagation Through Time (BPTT),

which uses an iterative chain rule on the unrolled computational graph. The BPTT algorithm on SNNs can be illustated by the figure fig. 1.14.

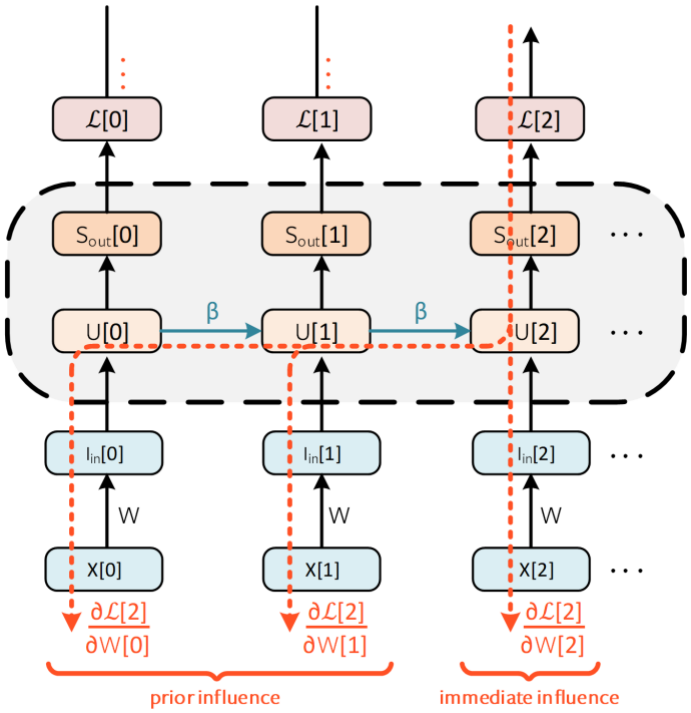

Figure 1.14: Backpropagation in the unrolled computational graph [20].

From the graph, it is easy to identify several possible pathways for BP, all contributing to the final loss, whether the neuron fire or not. The final loss is the sum of all losses over all time steps of an input:

$$\mathfrak{L} = \sum_t \mathfrak{L}(t) \tag{1.43}$$

Even though the parameters do not change in the forward path for the same input, they can contribute with multiple paths over time, not only the forward one. For example, at a single time step t the weight can contribute with its current value $W(t)$ and all its previous values $W(s < t)$ even if $W = W(0) = W(1) = ... = W(t)$.
As result:

$$\frac{d\mathfrak{L}}{dW} = \sum_t \frac{d\mathfrak{L}(t)}{dW(t)} = \sum_t \sum_{s=0}^{s=t} \frac{d\mathfrak{L}(t)}{dW(s)} \times \frac{dW(s)}{dW(t)} = \sum_t \sum_{s=0}^{t} \frac{d\mathfrak{L}(t)}{dW(s)} \tag{1.44}$$

$$\frac{d\mathfrak{L}}{db} = \sum_t \frac{d\mathfrak{L}(t)}{db(t)} = \sum_t \sum_{s=0}^{s=t} \frac{d\mathfrak{L}(t)}{db(s)} \times \frac{db(s)}{db(t)} = \sum_t \sum_{s=0}^{t} \frac{d\mathfrak{L}(t)}{db(s)} \tag{1.45}$$

The same goes for any other possible parameter. However, in all cases the loss derivative requires a passage by the membrane potential and Heaviside step function which are non-differentiable, to circumvent this issue we use surrogate gradient descent.

NOTES

- From a biological perspective, the BP equations do not seem biologically plausible [61]. But an interesting feature of BPTT is the analogy with biologically motivated rules, where the BP of the gradient over previous n time steps is proportional to β^n, which means an exponential scale of the gradient between different spikes, and this principle correlates with the STDP learning curves [61]. The figure fig. 1.15 shows the analogy between the effect of BPTT and STDP learning curve.

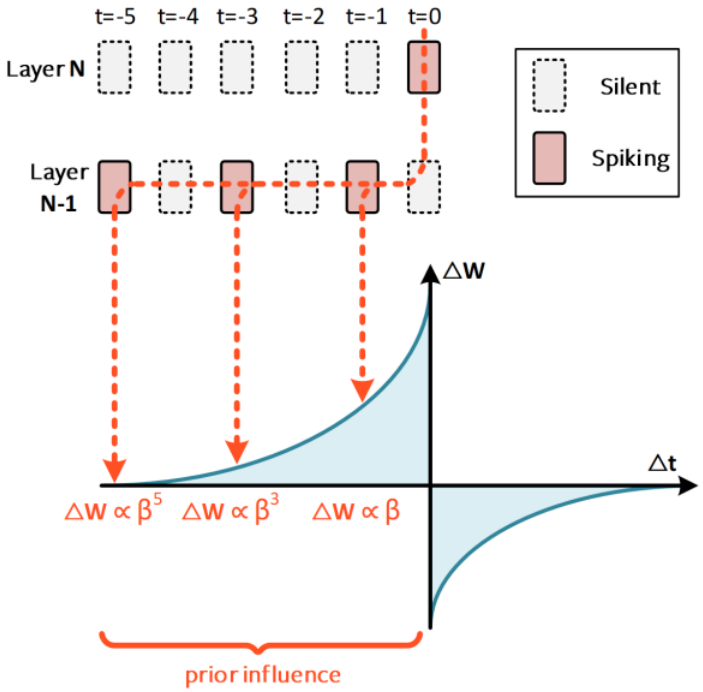

Figure 1.15: The effect of on synaptic link between neurons over time is very similar to the STDP principle.

- Unlike SpikeProp, BPTT is applicable for both rate and latency coding.
- Unlike ANNs, SNNs have more parameters, other than weigh and bias it also has the surrogate slope, threshold, and decay rate. Although it is always better off to set them as parameters, many investigations have shown that SNNs are pretty resilient to a fixed choice of those parameters and thus are considered hyperparameters [71, 61].

Yet, the spiking neuron is based on the Heaviside function, which derivative is the Dirac distribution, this can cause prevent the network learning, by canceling the impact of the backward derivative to zero, this problem is known as the dead neuron problem. To address these issues, state-of-the-art solutions are given by a surrogate gradient [49] that approximates the activation function and smooths its derivative in the backward pass. In other words, the forward pass preserves the conventional equations, but the backward pass requires the use of an approximate function instead of the Heaviside function. This allows a biased estimation of the gradient [20], even when the neuron is not spiking, bypassing the dead neuron problem, and results in significant performance improvement and even closing the gap with ANNs in several datasets such as MNIST and CIFAR10, and demonstrating improved performance on temporal tasks such as TIMIT speech recognition [7]. Various surrogate gradients can be used, and the choice can be treated as a hyperparameter. Examples of possible surrogate gradients in the figure fig. 1.16.

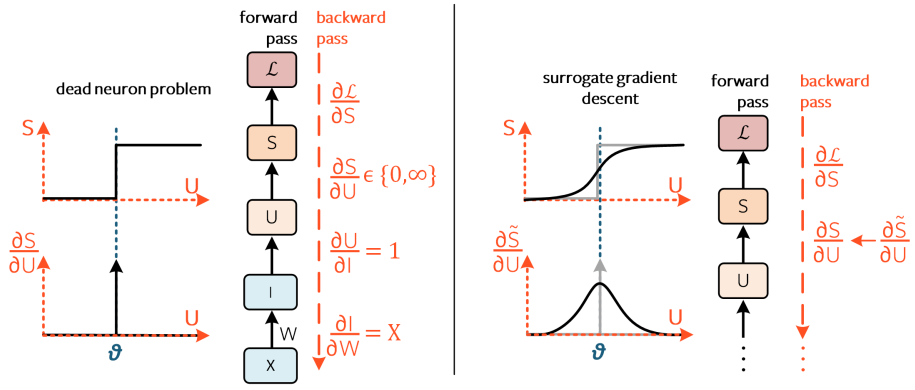

Figure 1.16: Difference between gradient and surrogate function [76].

- As a note of caution, the activation function includes two discontinuities, the Heaviside step function and the reset mechanism, and the surrogate only deals with the step function. The reset mechanism is not included in the backward pass using a surrogate function; empirically, it has been shown that including the reset mechanism in the computational graph would degrade network performance [71]. The membrane potential with the reset mechanism:

$$U(t) = \beta U(t-1) + WX(t) - S_{out}(t-1) \times \theta \tag{1.46}$$

$$= \beta U(t-1) + WX(t) - (\text{Surrogate}(U(t-1) - \theta)) \times \theta \tag{1.47}$$

  Without the reset mechanism (after detaching the reset mechanism):

$$U(t) = \beta U(t-1) + WX(t) \tag{1.48}$$

- The effect of the bias introduced by the surrogate is equivalent to random feedback on weight in the backward pass, the understanding of this approximation is limited [20].

Comparison between different learning rules

The following tables table 1.4 and table 1.6, explain a detailed comparison between different learning rules, advantages and drawbacks.

	ADVANTAGES	DRAWBACKS
CONVERSION	• Exploit already existing trained ANNs to create SNNs, without any prior training. • Low risk of dead neurons [20]. • Applicable on both supervised and unsupervised learning.	• Inefficient training process [20]. • Most conversion techniques use rate-codes, those are suboptimal in the usage of the time dimension [38], they require a large amount of time steps to achieve a sufficiently accurate approximation [38]. • None of the conversion techniques convert other sophisticated activation functions, except the ReLU. • Converting high precision values in each layer into spikes, offset the power/latency benefits initially sought from SNNs [20]. • Regardless how accurate is the conversion, the SNNs performance would always be limited by the original ANNs performance. • Fine-tuning with backpropagation degraded the network's performance, instead of improving it [51].
STDP	• This learning rule outperforms in unsupervised clustering. • Alleviates the need for computation of complex gradients. • Promotes competition among pre-synaptic neurons and shorter latencies [64].	• Lack of mathematical background. • Update only weights and do not consider other parameters.

Table 1.4: Comparison between different learning rules.

	Advantages	Drawbacks
SpikeProp	• Used with latency coding, where each neuron spike only once (per input). • Unbiased gradient calculation, since it doesn't need surrogate gradient.	• High risk of dead neurons. • Incompatible with rate coding. • Incompatible with dynamic datasets in which inputs have multiples frames, where the neuron is not encouraged to fire more than once. • Computationally costly.
BPTT	• Appliable for both latency and rate coding. • Consider all time steps in the gradient calculation. • Prevent dead neuron problem.	• Vanishing/exploding gradients. • Bias gradient estimation, since it require a surrogate gradient. • Computationally costly.

Table 1.6: Comparison between different learning rules.

1.2.3.3) *Objective function*

Also known as the loss function or the cost function, it is an estimation of the error, where the reduction of this loss thanks to the gradient descent algorithm promotes the right class. However, the encoding of the information is different using SNNs, which requires relatively different loss functions to quantify the error. There are two commonly used approaches in supervised machine learning that can be applied to different variables, both spike count and membrane potential, to encourage the right output to fire.

Negative Log Likelihood based loss functions: The loss is based on the logarithmic function, which is monotonic and preserves the variation with respect to the parameter, it is also convex with respect to its inputs (but not necessarily the parameters), as well as rapid convergence due to the properties of the logarithmic function in the interval $[0,1]$.

Least Error based loss functions: The classic square error calculation between the target value and the predicted value. Although it is convex with respect to its input, it is not necessarily convex with respect to the parameters. By minimizing this loss, the right class becomes closer to the right value.

Both approaches can be used for multilabel classification or multiclass classification, depending on the output activation function (Sigmoid or Softmax, respectively). Since we are mainly interested in multiclass classification, we assume that the output layer is the Softmax layer, as a result the loss functions used are Cross Entroppy (CE) and Mean Squared

Error (MSE). The following figure illustrate the different loss functions on the unrolled SNNs fig. 1.17. The tables table 1.8 and table 1.10 summarizes how these losses are used on rate and latency coding output respectively.

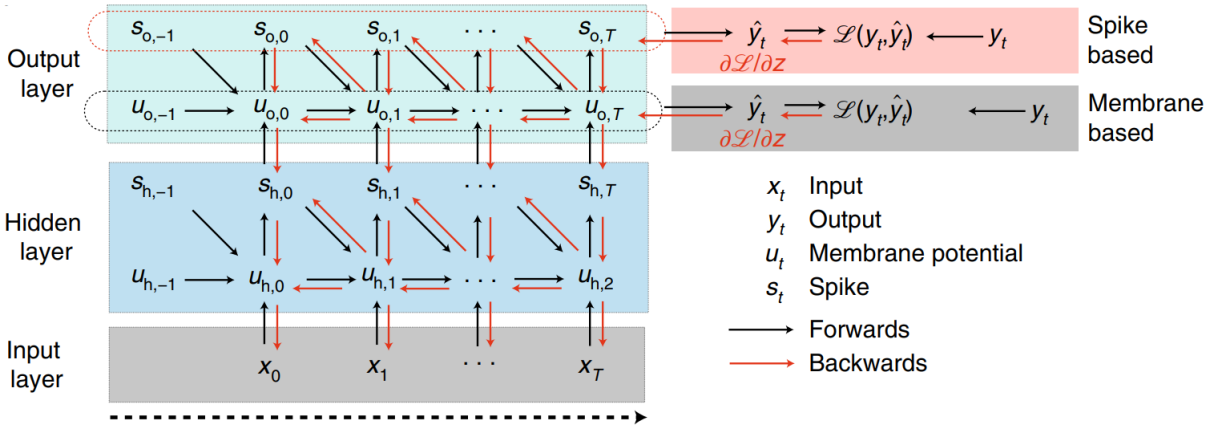

Figure 1.17: Roll-out of the computational graph of a spiking neuron as used for BPTT for a sequence $t = 0...T$ with different considering different loss function [69].

	CE	MSE
Spike count	**Over time steps:** Consider the spike sum (or average) of spike) $c_i = \sum_{t=0}^{T} s_t$ over time steps of each output node (N node) as Softmax logits. This function is noted Γ. $$\mathfrak{L} = \frac{1}{M}\sum_{k=1}^{M} p_{targeted_k} log(p_{predicted_k}) \quad (1.49)$$	**Over time steps:** Estimate the error as regression between the sum (or average) of spikes $p = \sum_{t=0}^{T} s_t$ and the targeted number of spikes. The target can be fixed as a percentage p times time steps $t = T \times p$ (or percentage), which means that we have pushed for a certain input to fire p of the time. This function is noted Δ. $$\mathfrak{L} = \frac{1}{M}\sum_{k=1}^{M} (c_{targeted_k} - c_{predicted_k})^2 \quad (1.50)$$ Alternatively, it can consider the spike count (sum of spike) $c_i = \sum s_j$ over time steps of each output node (N node) as Softmax logits, the loss function deals with probabilities where MSE is used as Brier score. $$\mathfrak{L} = \frac{1}{M}\sum_{k=1}^{M} (p_{targeted_k} - p_{predicted_k})^2 \quad (1.51)$$
	Each time step: Consider the spike at each time step of each output node as Softmax logits, and the loss is the average of each time step loss. This function is noted Υ. $$\mathfrak{L} = \frac{1}{M \times T}\sum_{k=1}^{M}\sum^{T} p_{targeted_k} log(p_{predicted_k}) \quad (1.52)$$ It is important to note that it is highly sensitive to the threshold value.	**Each time step:** The loss function is the average of each time step loss, time step loss is the square error between the output spike p and the targeted output value $t = \{0, 1\}$. $$\mathfrak{L} = \frac{1}{M \times T}\sum_{k=1}^{M}\sum^{T} (p - t)^2 \quad (1.53)$$
Membrane potential	**Over time steps:** Consider the sum (or average) membrane potential over time steps of each output node as Softmax logits that undergo the loss function.	**Over time steps:** Estimate the square error between the sum (or average) of scaled membrane potential of each output node, then calculating the loss as regression with predefined firing targets (or percentage) using MSE. Alternatively, it is possible to use Brier score. Alternatively, it is possible to use the maximum membrane across time steps as a logit. This function is noted Φ.
	Each time step: Consider the scaled membrane potential at each time step of each output node as Softmax logits. The loss function is the average of each time step loss. This function is noted Λ.	**Each time step:** Consider the membrane potential at each time step of each output node as Softmax logits, the objective is to maximize this value. The loss function is the sum of each time step loss. This function is noted Π.

Table 1.8: Rate coding loss functions.

	CE	MSE
SPIKE FIRING TIME	Consider the spike time of arrival, where the first neuron to spike is the predicted class. In order to use CE minimization, the arrival time should be converted with monotonically decreasing function to inverse value order as a low latency correspond to high value, only then a Softmax and CE loss function are used to predict the error.	It can be used with different ways: - Just like Brier score after pre-treatment of time arrival. - Using the conventional MSE by setting some target firing time to each output. - Also using the conventional MSE but regarding the relative spiking time between nodes, where the target class should arrive before other spikes by a fixed number of time steps.
MEMBRANE POTENTIAL	**Unreported in literature**, but is possible to use, since higher membrane potential in the target time steps promotes spiking, and lower ones makes it later.	**Over time step:** Likewise rate coding, higher membrane potential promote earlier spikes, the loss between the average of the membrane values overtime and the targeted membrane value using MSE. **Each time step:** The loss between the membrane values at each time step and the targeted membrane value using MSE, the final loss is the average of sub-loss of each time step.

Table 1.10: Latency coding loss functions.

NOTES

- Losses accumulated over time steps can be averaged by scaling the total number of time steps.
- Membrane based loss are highly sensitive to thresholds values, thus it requires scaling the membrane with respect to the threshold in order to reduce this sensivity.
- Unlike ANNs where hyperparameters only affect the convergence time, where network complexity and limit is directly related to its architecture. SNNs complexity and limit are highly sensitive to hyperparameters, where different loss functions expect different hyperparameters to converge, and this is more blatant when it comes to membrane-based loss functions that are highly sensitive to the threshold value. This difference can be justified by the assumption that the threshold, slope, and decay rate can be fixed as hyperparameters rather than parameters, where SNNs has proved a flexibility and robustness toward this assumption [71].

1.2.4) Regularization

Regularization is a tool to prevent overfitting problems by ensuring that the parameters have nondivergent values. SNNs share with ANNs parameters such as weights and biases;

thus the same regularization techniques of ANNs are applicable in SNNs, except for dropout and normalization. The dropout in SNNs is just like RNNs, it has to be spatially dependent [42]. Meanwhile, normalization cannot be applied to spikes directly, even though some research is used as it is [30], but there are other methods to circumvent this issue, such as:

- Normalization of the membrane potential [41].
- threshold-dependent Batch Normalization (tdBN) [75], where it normalizes the membrane potential with respect to the threshold before applying batch normalization.
- Batch Normalization Through Time (BNTT) [30], use the previous hidden state and the new one, in order to calculate the normalization parameters.

On the other hand, SNNs has a different configuration, which allows for other possible parameters such as decay rate, threshold, slope, etc. that need adapted regularization techniques. SNNs Regularization targets mainly the activity to establish sparsity occurrence, especially with rate coding schemes in the output layer, where the training process encourages spiking and increases the spike rate / membrane potential without an upper bound; this severely affects sparsity. Conventional techniques penalize excessive and insufficient spiking, but only up to a point. Other possible methods exist, such as:

- L1 / L2 layerwise regularization on the total number of spikes.
- An upper activity threshold, beyond the threshold, a penalty is applied using a refractory period for the neuron.

1.2.5) Training process challenges

The state-of-the-art machine learning model still outperforms SNNs in a variety of tasks ranging from image classification, time series prediction, language translation, and automatic speech recognition. This is mainly due to the advance techniques and model architecture, meanwhile, which SNNs have a narrower popularity and a lake of attention. This section focus on identifying SNNs drawbacks and issues that prevent from an appropriate learning process. Many of them are common between ANNs and SNNs which would helps SNNs to use deep learning frameworks to overcome it.

1.2.5.1) *Dead neuron*

When a neuron is unable to spike, this means that the neuron does not contribute to the loss. Furthermore, the loss derivation with respect to its parameters is null; this implies that the neuronal parameters cannot be learned nor updated. The issue is that the dead neuron is not able to learn or contribute, as if it does not exist, and it is very unlikely to get out of this state. At a large scale, it causes a lower network architecture complexity than it may seem. For instance, a network includes three layers of 256 neurons each, in total of 768 neurons where

80% of neurons are dead, which means that the network uses only about 154 neurons!
It can be due to:

- The nondifferentiable step function, which leads to a null gradient . To avoid this issue, a surrogate function allows a biased estimation of the gradient.
- A high threshold initialization, where most neurons are unable to achieve in the first stages of the training process and will never be able to spike [43]. The threshold initialization is empiric and is considered as hyperparameter tuning, if the threshold is a parameter, regularization helps to prevent this issue [43].

1.2.5.2) *Vanishing gradient*

This problem rises also in deep ANNs as well as SNNs, where the loss function is back-propagated from the output layers to the input layers, while pushing the error to zero in the first layers, a null loss can prevent the first layer from learning. For a network of N layers, the error at layer i can be written as:

$$\mathfrak{L}_i = \Pi_{j=N}^{i} W_j^T \times \mathfrak{L}_N \tag{1.54}$$

The propagation requires numerous weights multiplication in deep networks, where the weights are usually less than 1 adding to it the derivation of the activation function like sigmoid can be less than 1, which leads to almost null errors at the first layers in the network and cannot be updated. After some iterations, the networks would not be able to learn or progress. To avoid this issue:

- A high learning rate can make the gradient variation more impactful on the first layers, but also risk an exploding gradient at the later layers.
- An adequate initialization of weight parameters to maintain a certain balance, several initialization distributions exist, each can be adapted for a different context. In the case of the sigmoid activation function, a Xavier initialization is preferred [26]. However, for SNNs initialization methods, it is still an ongoing research topic [43].
- Normalization does not have a direct impact on the vanishing gradient, but makes the update more resilient to the parameter scale [32] and prevents small changes to the parameters from amplifying into larger and suboptimal changes in gradient. The normalization in the original paper [32] proposed to be used before the activation function, but experimental results proved that it has a better result if applied after the activation function. In the case of SNNs, normalizing the membrane potential gives very good results.
- Residual networks have recurrent connections between layers that help to skip the long sequential pathway, this creates a shorter path in deep networks that are less susceptible to the vanishing gradient.

- ReLU (or leaky ReLU) is also useful in practice, where its derivative does not contribute to the vanishing process, but cannot be applied in the context of SNNs.

1.2.5.3) *Exploding gradient*

Similarly, exploding gradient occurs when weights are important and more than 1, in this case it leads to an increase in the loss scale at the first layers, the important scaling causes a significant update to the parameters related to the first layers and overshot the global minima; it manifests itself by severe fluctuation in the learning curves. To prevent this issue:

- Gradient clip is a method that reduces the high gradient values to 1, the clip can be by cutting the gradient directions in which it exceeds 1, or by normalizing it is value to 1. Normalizing the gradient is preferred because cutting it may change the direction of the gradient.

- Orthogonal weights can be resilient to successive multiplication, yet it is still an active research topic for the case of SNNs [43], Xavier initialization still the best possible option [26].

- Likewise vanishing gradient, normalization helps in reducing the exploding gradient by rescaling the input, which leads to small and stable loss.

- Smaller learning rate, a big learning rate can risk overshooting the minima, thus higher error, when a higher error is backpropagated, it causes bigger update and maybe overshooting the minima again. Thus, it keeps raining in cyclic fluctuation without significant learning.

- Regularization L2 prevents high variation in parameters, which prevents exploding gradients.

- ReLU (or leaky ReLU) activation functions are also useful in practice, where the derivative is either 1 or 0 and does not contribute to the explosion of the loss, but cannot be applied in the context of SNNs.

- LSTM architecture is not a typical solution, but can help thanks to its forgot gate.

- Truncated backpropagation through time is a variant of BPTT, where forward and backward passes are run through chunks of sequences instead of the entire sequence.

1.2.5.4) *Overfitting / Underfitting*

SNNs just like their counterpart ANNs, can face over- and under-fitting issues, the prevention tools of ANNs can be applied to SNNs, as well as other approaches discussed in the previous section.

1.2.5.5) *Convergence time*

Simulating an SNN is a time-consuming process due to the additional temporal dimension of the signals. Not only that, but the simulation of LIF is sequential and time dependent to introduce the reset mechanism. Previous contributions [37, 27] tried to solve this issue by using a latency coding of 1 spike per neuron trained by SpikeProb, in this case there are no other following spikes that require a reset, and simulation is no longer sequential.

1.3) Benchmarking SNNs

Although SNNs are significantly different from ANNs, it is very important to highlight that they still share a common development pipeline including the performance evaluation procedure (generally using cross-validation), even if the benchmarking data sets can be different.

Multiple conversion techniques can be used to convert data formats, but none of them are accurate, since the neuromorphic camera has a temporal resolution of a few microseconds, while conventional cameras have a temporal resolution of tens of milliseconds. As a result, temporal frequencies are lost during frame-based sampling and cannot be reconstructed to accurately simulate neuromorphic algorithms [23], since the conversion creates only certain dynamics but doesn't contain any spatio-temporal patterns. Nevertheless, it is possible to use classical data sets for SNNs benchmarking with or without a conversion technique. A better way is using more adapted data sets that have been produced using neuromorphic cameras, such as DVS128, SHD, etc.

The evaluation metric generally depends on the task; The present work is interested in the proof of concept of the proposed idea on popular generalist datasets, where it mainly focuses on accuracy optimization. The choice of accuracy as a metric is biased by the SNNs community milestone.

1.4) Conclusion

This chapter has presented a review of the state-of-the-art literature on Spiking Neural Networks, differences from conventional Artificial Neural Networks, and how to use SNNs within the framework of RNNs, then we have concluded with the development challenges and issues. One particular problem of interest to us is to prevent dead neurons, since they drop performance and are hard to solve once they occur. To recap, our goal is to propose methods to overcome the current problems of SNNs in order to optimize their performances, and particularly accuracy. The next chapter will present the proposed approaches.

CHAPTER 2

PROPOSED APPROACHES

This chapter describes the three explored approaches in order to optimize SNNs performances, starting from their principles, then methodology and implementation techniques, and comparing with related work in terms of principle.

2.1) Negative threshold

2.1.1) Principle

Even though a detailed understanding of human sensory vision is not completely cracked, it has always been an important source of inspiration for human beings. Human vision physiology indicates interesting phenomena. Light excitation causes a series of inhibitions in photoreceptors, and these inhibitions can have the same effect of stimulating and excitation postsynaptic neurons [22].

In contrast, from SNNs perspective, negative modulated presynaptic inputs can never trigger a postsynaptic output regardless of how important the membrane potential value is. It constitutes an important loss of information. By setting a negative threshold, it is possible to communicate this information, improve the capacity of the model, enhance the SNNs performance as sequence. The figure fig. 2.1 illustrate the issue.

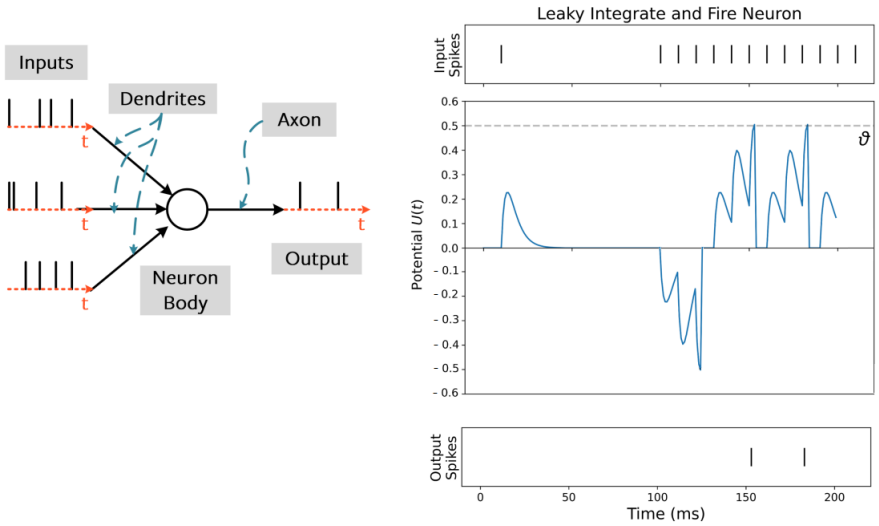

Figure 2.1: illustration of the loss information due to the non-presence of a negative threshold.

The importance of this work relies on important advantages, namely reducing information loss between layers in the forward pass, promoting learning and reducing the effect of dead neurons, as well as adding significant complexity while maintaining the same topology.

2.1.2) Methodology

From the previous equations, the output of the spiking neuron is:

$$U(t) = \beta U(t-1) + WX(t) - S_{out}(t-1) \times \theta$$

$$S_{out}(t) = H(U(t) - \theta)$$

Where H is the Heaviside step function. In our work, we propose replacing the thresholding function by O illustrated in the figure fig. 2.2.

$$O(t) = H(t) - H(-t)$$

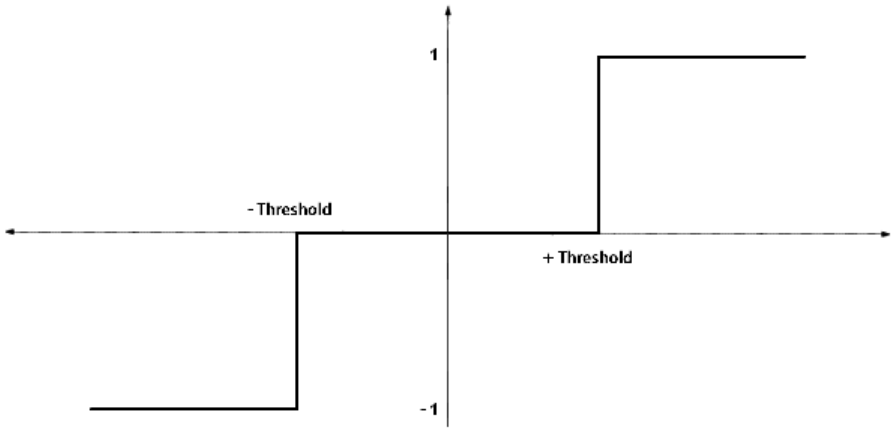

Figure 2.2: The new thresholding function.

The new function O enables negative inputs that are below the negative threshold to trigger a negative spike as desired. This new function requires a different surrogate in the backward pass. But it from a programming point of view, the new function can be expressed using Heaviside step function. Thanks to the modularity of the implemented packages, it is not necessary to redo the work from scratch.

NOTE

It is important to underlie that the output caused by the negative threshold is a negative spike. However, theoretically it is possible to use a positive spike, but empirically the result produced from this model shows a significant decrease in model performance, and the ability to learn this can be justified by the fact that using a positive spike creates less contrast and would make the model unable to distinguish between positive and negative inputs.

Related work

A previous work [70] investigated the possibility of using a double threshold, or even a multithreshold, to optimize the threshold-balancing [a] technique in the conversion process between ANNs and SNNs. **Though, using the negative threshold as part of network parameters using gradient-based learning has never been investigated before.**

2.2) Adaptive threshold

2.2.1) Principle

Although, most neurons share similar properties, different neurons in the brain have different parts, each of them is responsible for some functions which need some particular behaviors. A major difference between these parts is that each can have a different interval of thresholds. The threshold value also appears to be related to human genetics, experience, as well as many other factors such as sleep and medications [50]. This underlies the importance of the threshold value as parameters in the learning process, where it is usually considered a hyperparameter and set to a constant value during the learning process [71].

The fine line between sparsity and dead neurons is directly related to the threshold value. If the initialization threshold is relatively high, it can promote sparsity more than is needed and cause a lot of dead neurons, and once a neuron is dead, it is unlikely to restore it. Meanwhile, setting a very low threshold can cause significant loss and performance decrease, since most neurons would be highly sensitive and able to spike most of the time and require applying a regularization on the spike rates. The advantage of this work is that it uses backpropagation in order to identify the appropriate value of threshold that offer a good trade-off between sparsity and dead neuron. In our approach, we initialize thresholds at very low values to promote firing

[a] Threshold balancing is the process of identifying the optimal threshold that tradeoffs between sparsity, information loss and dead neuron.

among neurons regardless of the produced loss. This loss can be reduced using BP, and more sparsity can be produced by setting the threshold as a parameter during the learning process.

2.2.2) Methodology

To avoid complicated computation, instead of adapting the threshold, we can just consider using the bias as a threshold, since it has a similar effect, and we can consider linking the reset mechanism with respect to this bias. When the reset mechanism is detachable from the computational graph, the bias update value (gradient) is the exact conventional formula used in ANNs. By adding a bias b:

$$S_{out}(t) = H(U(t) + b - \theta)$$

$$\begin{aligned} U(t) &= \beta U(t-1) + WX(t) + b - S_{out} \times b \\ &= \beta U(t-1) + WX(t) + (1 - S_{out}) \times b \end{aligned}$$

Computation of the gradients is already ensured by conventional calculations of the bias and is implemented directly in the PyTorch framework.

Related work

This idea has been a subject of multiple previous works in which different methods have been suggested to optimize the threshold value. In practice, it can be by introducing an exponential decay [54]. The aim here is to use the threshold as regularization that prevent from unnecessary spikes, which means that once a neuron spike the threshold is reset to a high value to reduce the possibility of firing. It returns to its value using an exponential decay, this creates a refractory period. Another proposed alternative used with threshold balancing is conversion learning methods [15]. These propositions are less attractive compared to the bias in terms of computational complexity and effectiveness. It is important to note that many SNNs architectures already use an adaptive bias, but it does not have an impact on a threshold since it does not contribute as in the reset mechanism.

2.3) New loss function

2.3.1) Principle

In the conventional ANNs training process, the output prediction vector is expected to become closer to the target vector. For multiclass classification, the target output vector has a single target class labeled 1 and the others by 0, which generally works for ANNs. However, for SNNs this approach promotes the dead-neuron issue for non-target classes of the output.

Our method suggests changing the target vector probabilities. The objective is to encourage the other neuron rather than the target neuron to fire. By pushing them to fire, we create more

paths for gradient descent and reduce the probability of having dead neurons and vanishing gradients. From a machine learning point of view, this uncertainty does not affect the learning process; Since accuracy is estimated by calculating the number of correct estimations, a correct estimation is when the network assigns the higher probability to the targeted class (Winner-Take-All strategy), regardless of the value of the probability, either 100% or less.

2.3.2) Methodology

The Cross Entropy function is a special case of the Kullback-Leibler loss function. Kullback-Leibler divergence function is a statistical estimation of the divergence between two distributions [b]. The function describes how the predicted distribution P is different from the targeted distribution Q, from the perspective of the target distribution Q, if the divergence is null then both distributions are identical. Thus minimizing this function makes the distributions much closer to each other.

$$D_{KL}(Q||P) = H_p(q) - H(q) \tag{2.1}$$

Where:

$H(q)$: is the entropy function of the target distribution, where the entropy measures the uncertainty among a certain distribution of class C (of probabilities q).

$$H(q) = -\sum_{c=0}^{C} q(y_c) \times log(q(y_c))$$

$H_p(q)$: is the cross-entropy function between the target distribution (of probabilities q) and the predicted distribution (of probabilities p) between a certain C class of the distribution.

$$H_p(q) = -\sum_{c=0}^{C} q(y_c) \times log(p(y_c))$$

In the case of multiclass classification using conventional target vectors (a point from the distribution), where there is no uncertainty about the class, the entropy is null. In this case, the Kullback-Leibler loss is equal directly to the Cross Entropy loss.

$$H(q) = -1 \times log(1) - \lim_{q \to 0} \sum_{i=0}^{C-1} q \times log(q) = 0$$

$$D_{KL} = H_p(q)$$

[b] Statistical distance is a symmetric score that measure difference between statistical objects including distributions, yet this measure can be challenging as it can be difficult to interpret. Meanwhile, a divergence is not a symmetric measure, but it easily calculates how much a distribution differs from another, if it is null then the distributions are identical.

We note the new probability of the target class from the target distribution by P_0, the probability of the other classes is the value $\frac{1-P_0}{C-1}$. By changing the target vector, it results:

$$D_{KL} = -\sum_{c=0}^{C} q(y_c) \times log(p(y_c)) - \sum_{c=0}^{C} q(y_c) \times log(q(y_c)) \tag{2.2}$$

Where:

$$H(q) = -\sum_{c=0}^{C} q(y_c) \times log(q(y_c)) \tag{2.3}$$

$$= -P_0 \times log(P_0) - \sum_{c=0}^{C-1} \frac{1-P_0}{C-1} \times log\left(\frac{1-P_0}{C-1}\right) \tag{2.4}$$

$$= -P_0 \times log(P_0) - (1-P_0) \times log\left(\frac{1-P_0}{C-1}\right) \tag{2.5}$$

$$H_p(q) = -\sum_{c=0}^{C} q(y_c) log(p(y_c)) \tag{2.6}$$

$$= -P_0 \times log(p(y_C)) - \sum_{c=0}^{C-1} \frac{1-P_0}{C-1} \times log(p(y_c)) \tag{2.7}$$

Regardless of the new approximation of the target distribution, the entropy is no longer null. Nevertheless, the entropy is a constant, thus the entropy constant can be neither minimized nor deleted. Subsequently, the minimization concerns only the cross-entropy function:

$$D_{KL} + H(q) = H_p(q) \tag{2.8}$$

$$Min(D_{KL} + H(q)) = Min(D_{KL}) = Min\left(H_p(q)\right) \tag{2.9}$$

The estimated loss is for a single input from the data set of N input, which corresponds to the fact that the vector is a single point in the distribution. The total loss is the sum of all elementary losses; its minimization leads to reducing the distance between the point cloud of the target distribution and the predicted one:

$$\mathfrak{L} = \sum_{i=0}^{N} D_{KLi} \tag{2.10}$$

It is important to note that the Kullback-Leibler function is always convex when the target vector is fixed [34]. Also, the non-symmetry of the function is not an issue, as its minimization leads to 0, which means that the output and the target are identical [74]. Moreover, as with any other objective function, it can be applied to both the membrane potential and the spike rate.

Related work

This contribution has been inspired by this work [20], where they have proposed attributing an objective value to the non-targeted class to promote spiking and avoid the dead neuron problem; yet, their approach used a mean square error as the objective function and the value of the output layer was considered directly without any conversion into probabilities using a Softmax layer, so the problem is perceived as regression, not as classification.

The advantage of our contribution is that we consider the minimization of the divergence between two probability distributions, making it more adapted to multiclass classification and more appropriate than a MSE regression, where CE outperform the MSE function [9].

In addition, loss functions based on CE are usually preferred thanks to their logarithmic nature, which provided a faster convergence process. Although an appropriate empirical estimation is not possible since the convergence is highly sensitive to hyperparameters, a graphical mathematical illustration of the gradient can prove it as in the figure fig. 2.3.

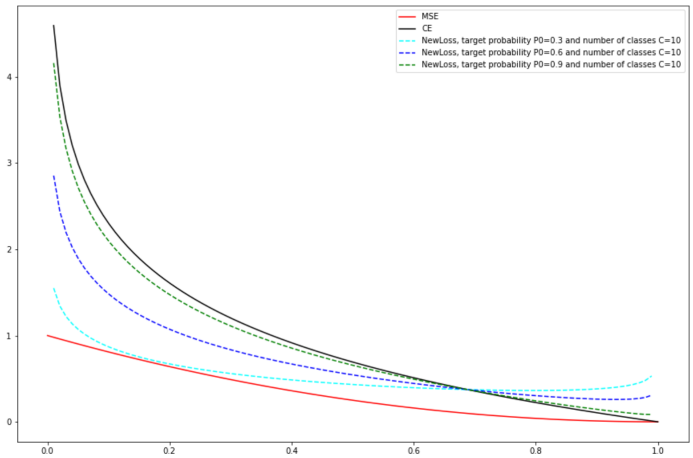

Figure 2.3: Plot of MSE, CE and the new CE with respect to inputs in range [0,1].

The convergence speed of the new loss function is directly dependent on the target class probability P_0.

NOTES

- To estimate the amount of noise introduced by the new approximation, we can measure the distance between the original distribution of the target and the new assumed distribution.
- We do not have an estimate of the optimal value of the new probability P_0 affected to the target class, but we can find a minimum limit.

$$\frac{1}{C} < P_0$$

- The probability sum of the non target classes is $1-P_0$. Intuitively, each non target class can be set a target of $\frac{1-P_0}{C-1}$.

Part II:

Experimental part

> Fairy tales are more than true: not because they tell us that dragons exist, but because they tell us dragons can be beaten.
>
> C.K. Chesterton

CHAPTER 3

EXPERIMENTAL PROTOCOL

This chapter deals with the evaluation protocol, starting with a presentation of the hardware tools and software frameworks needed. Then, we present the used data set with a brief analysis of the data and proposed pre-processing, discuss the evaluation strategy and the metric used. Finally, it exposes the results from each approach and compares with prior work.

3.1) Hardware tools

We have used the Google Colaboratory service, known as Google Colab. It is a Google Research cloud service that allows online access to GPUs using an interface based on a Jupyter notebook. Highly adapted for high-calculation and tensor-based frameworks, making it a powerful computing tool for machine learning. It contains other upgrade offers with higher resources, such as Colab Pro and Colab Pro +. For our computation, we have used a Colab Pro + service, that offer the resources summarized in the following figure fig. 3.1.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   41C    P0    27W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 3.1: Summary of the offered resources.

3.2) Frameworks and Libraries

A deep learning framework can include multiple libraries, APIs, compilers, and other tools that allow rapid, easy, and efficient development without going into the details of the underlying algorithms. Provides a clear and concise way of defining models using a collection of pre-built and optimized components.

3.2.1) Python

Python is an open-source, interpreted, high-level, object-oriented programming language. It is designed to be used on multiple platforms, for simple or complex programs, and has several libraries useful for programming in different contexts and applications, Python is becoming the default choice and the most used in the field of Machine Learning and AI. We use Python version 3.9.1 for the implementation of all our methods.

3.2.2) PyTorch

Machine learning Python open source framework, developed primarily by the Facebook AI Research lab (FAIR) in 2016. It is commonly used in research thanks to it is flexibility, modularity, and parallelism, which ensures that any new neural network architecture can be easily generalized and implemented. But the main reason behind this choice is that PyTorch is widely adopted by the research community and most SNNs packages are PyTorch based.

3.2.3) SnnTorch

SnnTorch is a Python package for performing gradient-based learning with SNNs. It extends the capabilities of PyTorch, taking advantage of its GPU accelerated tensor computation and applying it to spiking networks.

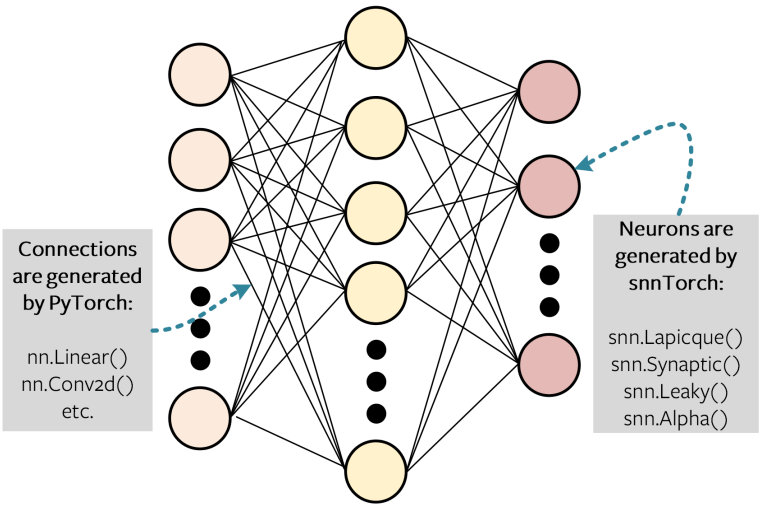

Figure 3.2: Layers supported by SnnTorch and compatibility with PyTorch [20].

3.2.4) Optuna

An open source framework for optimized automated hyperparameter search, it allows using advance Bayesian optimization techniques instead of random and manual research, it provides the 4 sampling algorithms, and stops unpromising trials at the early stages. It can be easily plugged into the code using a predefined template.

3.2.5) NumPy

NumPy is an open-source Python library and a very powerful tool in scientific computing. It is based on multidimensional arrays through matrix representation and functions that operate on arrays. The array object called ndarray is up to 50x faster than traditional Python list, where most low-level operations are written in C.

3.2.6) Matplotlib

Matplotlib is a Python plotting library that can be used to create 2-D plots that can be saved as figures. Its core module is called PyPlot which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the advantage of being free and open source.

3.3) Datasets

We evaluated our work in the context of supervised training using backpropagation on two main datasets, F-MNIST and DVS128 gesture, both commonly used by the SNNs community for benchmarking.

3.3.1) FMNIST

Although SNNs are not adapted for static datasets, it is important that SNNs are able to perform well on these types of data sets for practical reasons, since neuromorphic sensors still very expensive and not yet very popular, it is unlikely to become popular soon. Fashion-MNIST is a static data set provided by Zalando [67], it contains 28x28 grayscale static images of 10 categories of fashion products, and provides a split Train / Test consisting of 60000 samples for training, and 10000 for testing. We prefer to use this data set over MNIST, because MNIST is already saturated, while F-MNIST is still a challenging task for SNNs. The figure fig. 3.3 includes some sample examples from the F-MNIST data set.

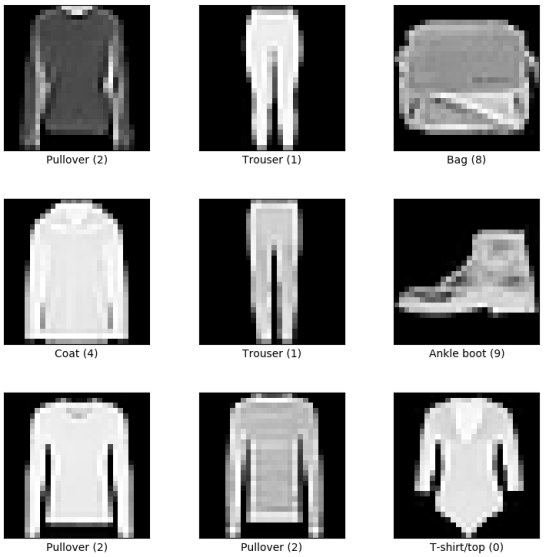

Figure 3.3: Samples from F-MNIST dataset.

3.3.2) DVS128 Gesture

More sophisticated and dynamic data set is recorded for benchmarking using a dynamical vision sensor, such as DVS128 Gesture. DVS128 recorded by IBM [67, 3], for hand action recognition of 29 participants under different illumination conditions, there are 10 categories of actions and a category for other random actions, it is divided into 1076 training samples (approximately 2h) and 288 test samples (approximately 1960s), sample durations may vary between $1s$ to $18s$ [35], but the average is about $6.5 \pm 1.7s$ [39]. It is important to note that the neuromorphic camera Dynamic Vision Sensor (DVS) records the input into two channels, one for the increase of the luminance (on the spikes) and the other for the decrease of the luminance (off-spikes); this adds more dynamics and illustrates more the notion of time in the sample, which is a series of frames. The figure fig. 3.4 includes some sample examples from the F-MNIST data set.

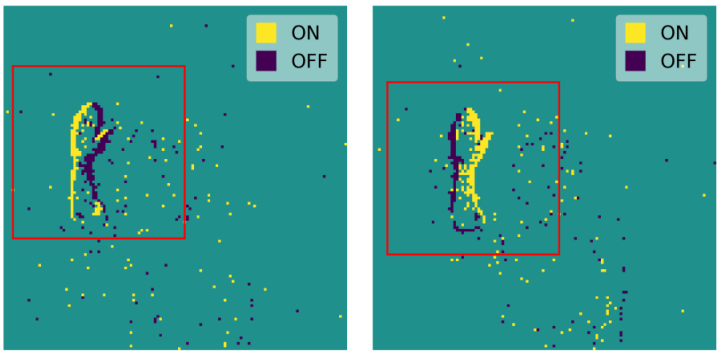

Figure 3.4: Two samples from DVS128 dataset, from categories Arm circling clockwise and Arm circling anticlockwise (left and right respectively) [35].

3.4) Exploratory Data Analysis

Exploratory Data Analysis refers by definition to the critical process of performing initial investigations on data to discover patterns, spot anomalies, test hypothesis, and verify assumptions with the help of statistical summary and graphical representations.

3.4.1) F-MNIST

For the F-MNIST, there is no necessary balance plot, where the sample distribution is uniform, each class has exactly 6000 samples for training and 1000 samples for testing.

3.4.2) DVS128 Gesture

For the DVS128 Gesture, the data balance is illustrated in the next table table 3.2:

LABEL	TRAINING	TEST	DESCRIPTION
1	97	24	Clapping
2	98	24	Right-hand waving
3	98	24	Left-hand waving
4	98	24	Right arm circling clockwise
5	98	24	Right arm circling anticlockwise
6	98	24	Left arm circling clockwise
7	99	24	Left arm circling anticlockwise
8	196	48	Arms rolling
9	98	24	Air drum
10	98	24	Air guitar
11	98	24	Other gestures

Table 3.2: Labels and samples amount of all the different classes in the Dvs128 Gesture dataset and their description.

The number of samples of label 8 is doubled, since arms rolling were recorded and labeled with 8 for both rotation directions. This creates some imbalance in the data to circumvent the issue, two possible approaches can be used, the data preprocessing approach by oversampling with augmentation or under-sampling, or the algorithmic approach by weighting the loss. However, our data are a sequence of frames and do not fit the available pre-implemented function for data augmentation. Therefore, data augmentation is not an option; cutting off excess samples is also not an option, since the data are already very small. As a solution, we opted to add weights to the loss function to re-scale the balance.

Another interesting feature about this dataset is its time dependence, where each sample is a series of frames and the lengths are not uniform. It is important to have an idea regarding the average sample duration, since it helps to establish a uniform attention window instead of considering the totality of frames, it has been shown to improve the learning process, training time, and model performance [35]. Here is a plot fig. 3.5 to illustrate the average characteristics.

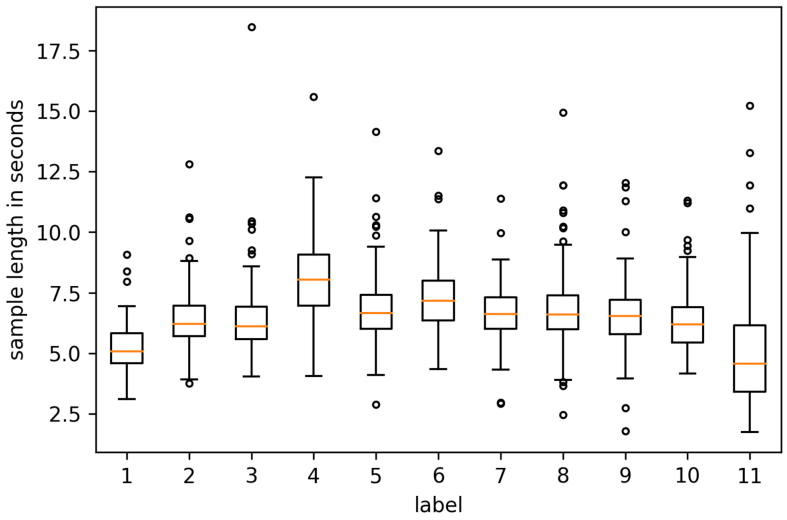

Figure 3.5: Statistics on sample duration for each label in the Dvs128 Gesture dataset [35].

The red horizontal lines represent the median sample duration, each box indicates the interquartile[a] range, while the circles constitute the outliers.

3.5) Data preprocessing

Data preprocessing aims to convert the available data to useful and compatible ones, it includes data cleaning (in case of missing or mislabeled values), data transformation (in order to reduce features or simply it), and all operations that need to be applied before it has been used in the network.

3.5.1) F-MNIST

- No data augmentation is necessary, since the datasets are already large enough.
- The data set is transformed into tensors and normalized.
- Both train and test sets are merged, then randomly divided into train / validation / test with proportions of 60%, 20%, 20% respectively.
- Each set is batched with 128 samples in each batch.
- In all sets, the last batch is discarded if it is incomplete.
- The shuffle option is activated for the train loader to reduce the overfit effect. The shuffle does not have an impact on the validation/test sets, so it is turned off.

[a] Quartiles are three values that separate a set of data placed in ascending order into four subsets with exactly the same number of data.

3.5.2) DVS128 Gesture

- The data set is small and unbalanced, but data augmentation is not possible due to the lack of the necessary tools. Many publications used a conversion of the event into a single frame by averaging total frames; it is a useful trick to enable data augmentation, but dropout time-varying properties of the dataset. Data reduction for balancing the data is also not possible since the dataset is not large enough.

- The data are resized to 32x32 using a pooling function by setting a rescaling factor to 4.

- The attention window is set as the average time of all classes, which is 6s, this value is obtained from previous work [35, 56] that investigated the impact of window size in DVS128. The samples have different length, if the sample is shorter than the attention window in this case it can be extended by padding and masking. Otherwise, it is truncated to fit the window.

- Event-based frames are too sparse to extract features, even for humans. Thus, we generally integrate events into multiple frames [44]. The number of integrated time stamps in microseconds is set to 3000 sequences per frame. The choice of this value is very important because it defines the number of stamps merged to constitute a frame; when this number is low, the frame is not significant and not comprehensible even by humans, when the number is high, it can produce many blurs, the figure illustrates the problem fig. 3.6. Our choice was based on experimental results and previous work [17].

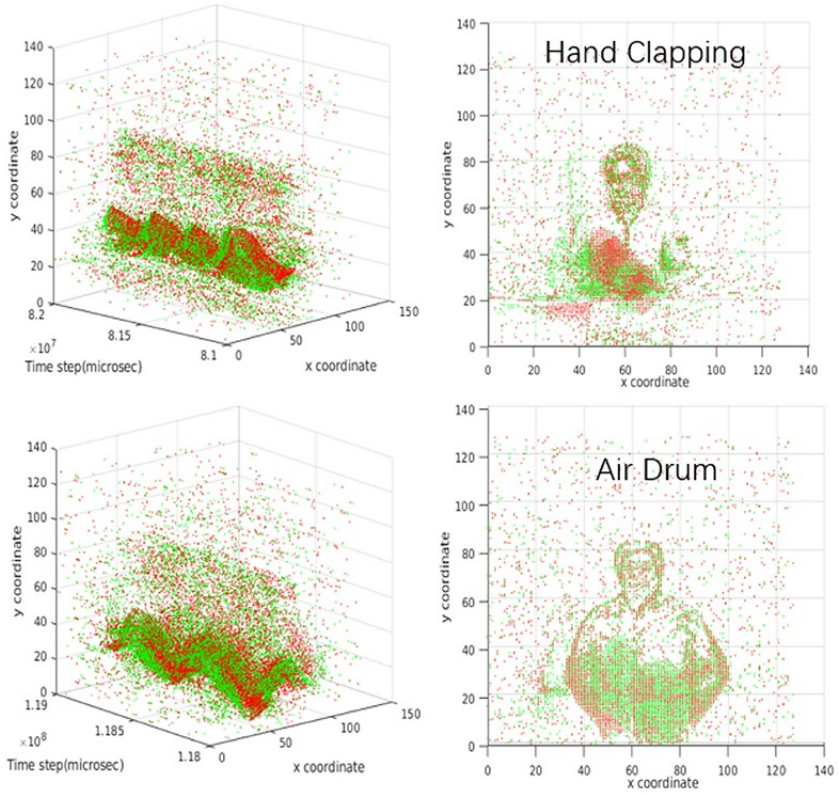

Figure 3.6: Example of integration into frames from DVS128 Gesture.

- Both train and test sets are merged, then randomly divided into train / validation / test with proportions of 80%, 10%, 10% respectively.
- Each set is batched with 128 samples in each batch; larger batch is not an option due to the hardware constraint (GPU Ram).
- In all sets, the last batch is dropped out if it is incomplete.
- The shuffle option is activated for the train loader to reduce the overfit effect. The shuffle does not have an impact on the validation/test sets, so it is turned off.

3.6) Evaluation strategy

3.6.1) Evaluation metric

As mentioned earlier, the main interest of the community is to close the accuracy gap between SNNs and ANNs. Therefore, we target accuracy as the main metric, but in the case of DVS128 Gesture (unbalance data set) we suggest also using the confusion matrix.

3.6.2) Baseline architectures

For an accurate evaluation process, we suggest setting baseline models for each data set. First, we optimize hyperparameters of models and evaluate the performance of these reference models. Then, once the models are set, we add the suggested configurations of each approach, and then we re-evaluate performances.

Different datasets involved have different complexity, which requires different architectures. All in all, we use mainly convolution-based architectures and linear output layer, and the spiking layers are used in between since they have the effect of an activation function. The architecture is fixed using experimental results. However, it was difficult to implement a larger architecture, as we are limited by training time and resources.

3.6.2.1) *F-MNIST*

We have used an architecture of 2 stages of convolution layers, each followed by a 2x2 max-pooling layer, without padding or stride, and finally a dense layer of 10 output. The choice of convolutions is trivial, since they have shown their ability to deal with image classification in the past decade. The pooling significantly reduces irrelevant features, the max pooling is chosen because the data set is grayscale images with a black background (0), where it outperforms any other pooling method [21]. The number of filters, channels, and parameters are experimentally fixed. The architecture is illustrated in the figure fig. 3.7.

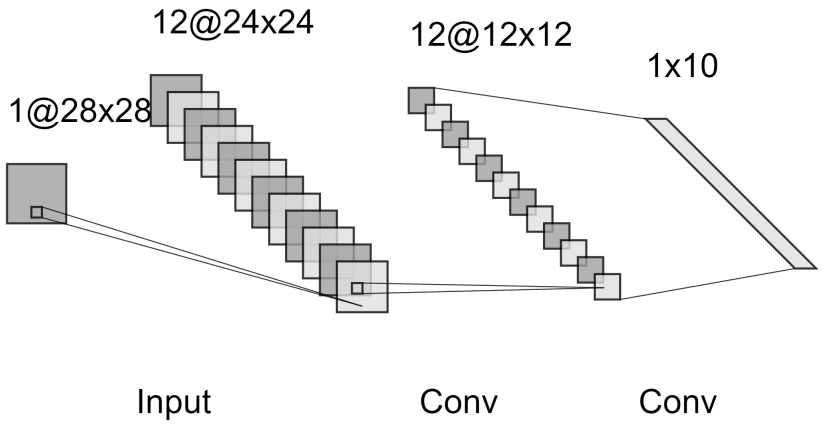

Figure 3.7: Architecture used on the F-MNIST dataset.

This architecture uses the following configuration / optimization techniques during the learning process:

- The surrogate function is the Fast Sigmoid with a slope $k = 25$.
- The decay rate of the hidden state $\beta = 0.95$.
- The optimizer is Adam, the learning rate varies across different loss functions, with no weight decay.
- No regularization L1/L2 is used.
- The dropout rate (Dropout2d and Dropout) varies across different loss functions, it is set between 0.01 and a maximum 0.1 (higher rates in the first layers and no dropout at the output layer).
- We use batch normalization for on the spike output.
- The output layer uses rate-coding and rate-based loss functions.
- After each epoch, the network produced is saved using checkpoints.

3.6.2.2) *DVS128 Gesture*

We have used an architecture of 3 stages of convolution layers, each followed by a 2x2 max-pooling layer, without padding or stride, and finally a dense layer of 10 output. The choice of convolutions is trivial, since they have shown their ability to deal with image classification in the past decade. The pooling significantly reduces irrelevant features, the max pooling is chosen because the data set is grayscale images with a black background (0), where it outperforms any other pooling method [21]. The number of filters, channels, and parameters are experimentally fixed and inspired by previous work [19, 56]. The architecture is illustrated in the figure fig. 3.8.

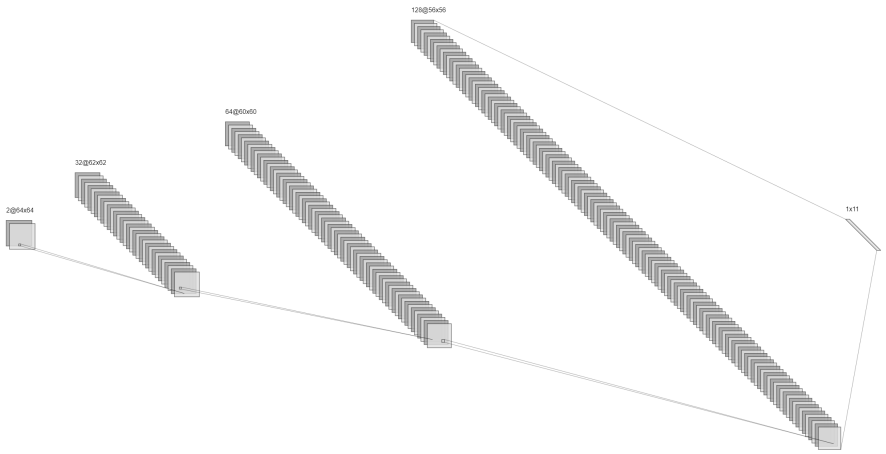

Figure 3.8: Architecture used on the DVS128 Gesture dataset.

This architecture uses the following configuration / optimization techniques during the learning process:

- The surrogate function is a Fast Sigmoid with varying slope.
- The decay rate of the hidden state $\beta = 0.72$ and $\beta = 0.97$.
- The optimizer is Adam, learning rate varies across different loss function.
- A regularization L2 is not used, but the weight decay of the optimizer has a similar effect, the decay value across different loss functions.
- The Cosine annealing scheduler is set with a frequency between 50 to 150.
- A gradient clip by norm to 1 for avoiding exploding gradients.
- Initialize weight using Xavier distribution to prevent vanishing gradients.
- A spatial dropout (Dropout2d and Dropout), is applied since SNNs are recurrent networks, the rate varies across layers (higher rates in the first layer and no dropout in the output layer) of the network and across different loss functions.
- Batch normalization is applied to both membrane potential and spikes.
- The output layer uses the rate coding and rate based loss functions.
- After each epoch, the network produced is saved using checkpoints.

CHAPTER 4

EXPERIMENTAL RESULTS

This chapter exposes the experimental results and the evaluation of each proposed approach on different datasets. In order to appreciate and assess the improvement brought thanks to our methods, we compare optimized models with the baseline models. Then we analyze and discuss results. Finally, we select the most performing model in each dataset to compare them with state-of-the-art accuracies.

4.1) Optimized models vs Baseline models

4.1.1) Negative threshold

The tables table 4.2 and table 4.4 are accuracy (Train/Validation/Test) of the baseline models and optimized models on F-MNIST dataset and DVS128 Gesture dataset.

LOSS FUNCTION	BASELINE			RESULTS		
CE SPIKE RATE Γ	91.55%	89%	87.2%	92.99%	89.11%	**89.16%**
CE SPIKE RATE Υ	92.16%	88.33%	**88.23%**	89.98%	88.15%	87.81%
MSE SPIKE Δ	90.98%	88.04%	88.05%	91.41%	88.41%	**88.16%**
CE MEMBRANE Λ	80.21%	80.06%	80.41%	87.35%	87.30%	**86.87%**
CE MEMBRANE MAX Φ	88.55%	85.71%	**85.82%**	85.01%	84.84%	84.76%
MSE MEMBRANE Π	88.13%	85.82%	85.20%	92.55%	90.03%	**90.04%**

Table 4.2: Comparison between first approach results (Train/Validation/Test) and baseline for the F-MNIST data set.

Loss function	Baseline			Results		
CE Spike Rate Γ	94.44%	93.75%	93.55%	-	-	-
CE Spike Rate Υ	91.84%	91.41%	90.66%	92.53%	91.41%	**91.15%**
MSE Spike Rate Δ	91.75%	90.62%	89.84%	-	-	-
CE Membrane Λ	92%	91.41%	90.12%	95.96%	92.19%	**91.07%**
CE Membrane Max	91.93%	88.28%	87.67%	94.10%	93.75%	**93.4%**
MSE Membrane Π	76.11%	75.41%	74.76%	82.99%	78.91%	**76.22%**

Table 4.4: Comparison between first approach results (Train/Validation/Test) and baseline for the DVS128 Gesture data set.

4.1.2) Adaptive threshold

Similarly, the table 4.6 is accuracy results (Train/Validation/Test) of the baseline models and optimized model on the F-MNIST dataset. Due to lack of time, it was not possible to test on the DVS128 Gesture dataset.

Loss function	Baseline			Results		
CE Spike Rate Γ	91.55%	89%	87.2%	91.94%	88.86%	**88.97%**
CE Spike Rate Υ	92.16%	88.33%	88.23%	92.19%	89.08%	**88.68%**
MSE Spike Δ	90.98%	88.04%	88.05%	91.41%	88.41%	**88.16%**
CE Membrane Λ	80.21%	80.06%	80.41%	85.53%	84.03%	**84.20%**
CE Membrane Max Φ	88.55%	85.71%	**85.82%**	86.99%	85.80%	85.13
MSE Membrane Π	88.13%	85.82%	85.20%	86.99%	85.80%	**85.66%**

Table 4.6: Comparison between second approach results (Train/Validation/Test) and baseline for the F-MNIST data set.

4.1.3) New loss function

The tables section 4.1.3 and section 4.1.3 are accuracy (Train/Validation/Test) of the baseline model using the new loss function compated to other loss functions on F-MNIST and DVS128 Gesture Datasets respectively.

	Loss function	Baseline		
Conventional	CE Spike Rate Γ	91.94%	88.86%	88.97%
	CE Spike Rate Υ	92.19%	89.08%	88.68%
	MSE Spike Δ	91.41%	88.41%	88.16%
	CE Membrane Λ	85.53%	84.03%	84.20%
	CE Membrane Max Φ	88.55%	85.71%	85.82%
	MSE Membrane Π	86.99%	85.80%	85.66%
New	CE Membrane Ω	91.38%	89.82%	**89.07%**
	CE Spike Rate Ψ	91.68%	89.60%	**88.8%**
	CE Spike Rate Θ	90.80%	88.78%	**88.50%**

Table 4.8: Comparison between third approach results (Train/Validation/Test) and baseline for the F-MNIST data set.

	Loss function	Baseline		
Conventional	CE Spike Rate Γ	94.44%	93.75%	93.55%
	CE Spike Rate Υ	91.84%	91.41%	90.66%
	MSE Spike Δ	90.89%	90.62%	89.66%
	CE Membrane Λ	92%	91.41%	90.12%
	CE Membrane Max Φ	91.93%	88.28%	87.67%
	MSE Membrane Π	86.11%	85.41%	84.76%
New	CE Membrane Ω	94.99%	94.53%	**94.19%**
	CE Spike Rate Ψ	93.49%	92.19%	**93.75%**
	CE Spike Rate Θ	94.18%	95.88%	**95.53%**

Table 4.10: Comparison between third approach results (Train/Validation/Test) and baseline for the DVS128 Gesture data set.

The loss can be applied to the membrane potential and the sum or spike, either at each time step or on the average over time steps. The new loss is applied on average spike rate over time steps noted Θ, spike rate at each time steps noted Ψ, and the membrane potential at each time step noted Ω.

4.2) Results analysis & Discussion

- Networks have been well-trained, the models do not suffer from either bias or variance.
- Networks trained with the proposed approaches generally outperform in most cases; even when it does not, the results are pretty close.
- We notice that networks perform better on the DVS128 Gesture more than the F-MNIST, we argue that the DVS 128 Gesture contains spatio-temporal patterns that are not included in the F-MNIST, which is more meaningful features and compatible with SNNs.
- We notice that using different loss functions can produce different results. This can be explained by two reasons, different loss functions have different curve shapes, and the network performance is not only related to the architecture but also the SNNs parameters that are assumed as hyperparameters (threshold, decay rate, slope). We have also observed that this impact is not only on the training accuracy, but also on the convergence time. Therefore, the most appropriate way to compare the results is only after setting all these parameters as learnable. More extensive analysis should be done to explore the reasons behind these results, till now there is not a prior work that compare different loss functions impact on the training and complexity.
- We also notice that spike based loss outperform membrane based loss functions, this can be due to the high sensitivity of the membrane potential to the threshold values.
- It is important to highlight that the combination of several methods can produce better results. For example, the combination of the negative threshold method and the new CE

for the membrane produced better result in the F-MNIST where it reached 91.13% of accuracy.

- It is important to highlight that SNNs networks are highly sensitive to hyperparameters, especially using dynamical data sets. Where training a single network may require a lot of time to figure out the right parameters.
- As mentioned earlier, accuracy is the primary metric, yet in the case of DVS128 Gesture where the data are imbalanced, accuracy can be misleading; Thus, it is possible to use the confusion matrix to have a more accurate evaluation, we use it only on our best model fig. 4.1.

4.3) Comparison with prior work

From previous results, we take only the most performing networks to compare them with prior work. For the F-MNIST data set we use the negative threshold approach, and for the DVS128 Gesture we use the new loss function approach.

4.3.1) Comparison & Discussion for F-MNIST

The following section 4.3.1 summaries the state-of-the-art models that have good performances on the F-MNIST dataset. For each model we describe the architecture, number of parameters, if the model process the F-MNIST directly or after processing (conversion to dynamical data set), and finally the accuracy that has been using the model achieved.

METHOD	ARCHITECTURE	PARAM	CONV	ACCURACY
SNN-Tha [19]	16C5-AP2-64C5-AP2-10	(~6599)	✗	87.92%
BSNN [19]	16C5-AP2-64C5-AP2-10	(~6599)	✗	86.95%
ST-RSBP [72]	400-R400-10	478841	✓	90.13%
LISNN [16]	32C3-AP2-32C3-AP2-10	(~11575)	✗	**92.07**%
Our work	12C5-MP2-32C5-MP2-10	5355	✗	90.44%

Table 4.12: Comparison of accuracies performance of different methods for the F-MNIST data set.

Most prior paper avoid converting static dataset, because the produced results do not include any spatio-temporal patterns, and require an important convergence time. The drawback of using the static dataset directly is the energy loss at the input layer, and we do not exploit SNNs properties.

The results obtained from our work do not outperform the current state of the art, it may be due to unadapted choice of architecture (number of parameter is very limited comparing to other architectures). However, it is still very competitive.

4.3.2) Comparison & Discussion for DVS128 Gesture

The following section 4.3.2 summaries the state-of-the-art models that have good performances on the DVS128 Gesture dataset. For each model we describe the architecture, number of parameters, number of class considered (some models drop the outliers class), and finally the accuracy that has been using the model achieved.

METHOD	ARCHITECTURE	PARAM	CLASSES	ACCURACY
SNN-T [19]	16C5-AP2-32C5-AP2-11	(~15303)	11	91.32%
BSNN [19]	16C5-AP2-32C5-AP2-11	(~15303)	11	88.54%
TrueNorth [3]	16 layers CNN based	-	11	91.77%
SLAYER [56]	8 layers CNN based (Not detailed)	-	10	93.64%
SCRNN [68]	32C5-MP2-64C3-MP2-128C3-MP2-512-11	662000	11	92.01%
SSNN [17]	4C5-8C5-8C3-16C3-10	14000	10	92.01%
Our work	16C5-MP2-32C3-MP2-64C7-MP2-11	115148	11	**95.53%**

Table 4.14: Comparison of the accuracy performance of different methods for the DVS128 Gesture data set.

First, we notice that our work outperform the SLAYER based CNN network with 1.89% of accuracy with a significantly smaller architecture (we can not estimate since the paper did not detail the architecture). However, it is worth mentioning that SLAYER use a larger architecture of 8 CNN filters, does not rescale input, and removes the outlier class (11) as it is a challenging class and reduces network performance, which allows better results. It is also necessary to mention that SLAYER requires 739 epochs to achieve those results [56], meanwhile none of our networks exceed 200 epochs.

Also, it should be taken into consideration that all previous results can be even inaccurate since none of them used the Train/Validation/Test split, and none of them tried to balance the DVS128 data nor used prevention technique or have tried other metrics with DVS128 Gesture data set (except TrueNorth [3]). Thus, those accuracies can be very misleading.

In addition, most of the previous results have been produced using advanced hyperparameter research techniques instead of random search, those tools (already implemented in the Optuna and RayTune frameworks) may require on average between 3 and 5 days to find the optimum parameters, and a such solution cannot be implemented for all our models due to the lack of time and computational resources. As a result, our contribution can have better results and be more promising.

Finally, for an accurate evaluation, we use the confusion matrix in our model; the results are plotted in the figure fig. 4.1. From the plot, we can say that the model have very good results on most classes, even the random one. However, the 'Clapping' class and the 'Air drum' are very confusing even for human beings, this has been deducted from our experimental visualization, and mention in other papers [68], which explains the performance's drop on those classes.

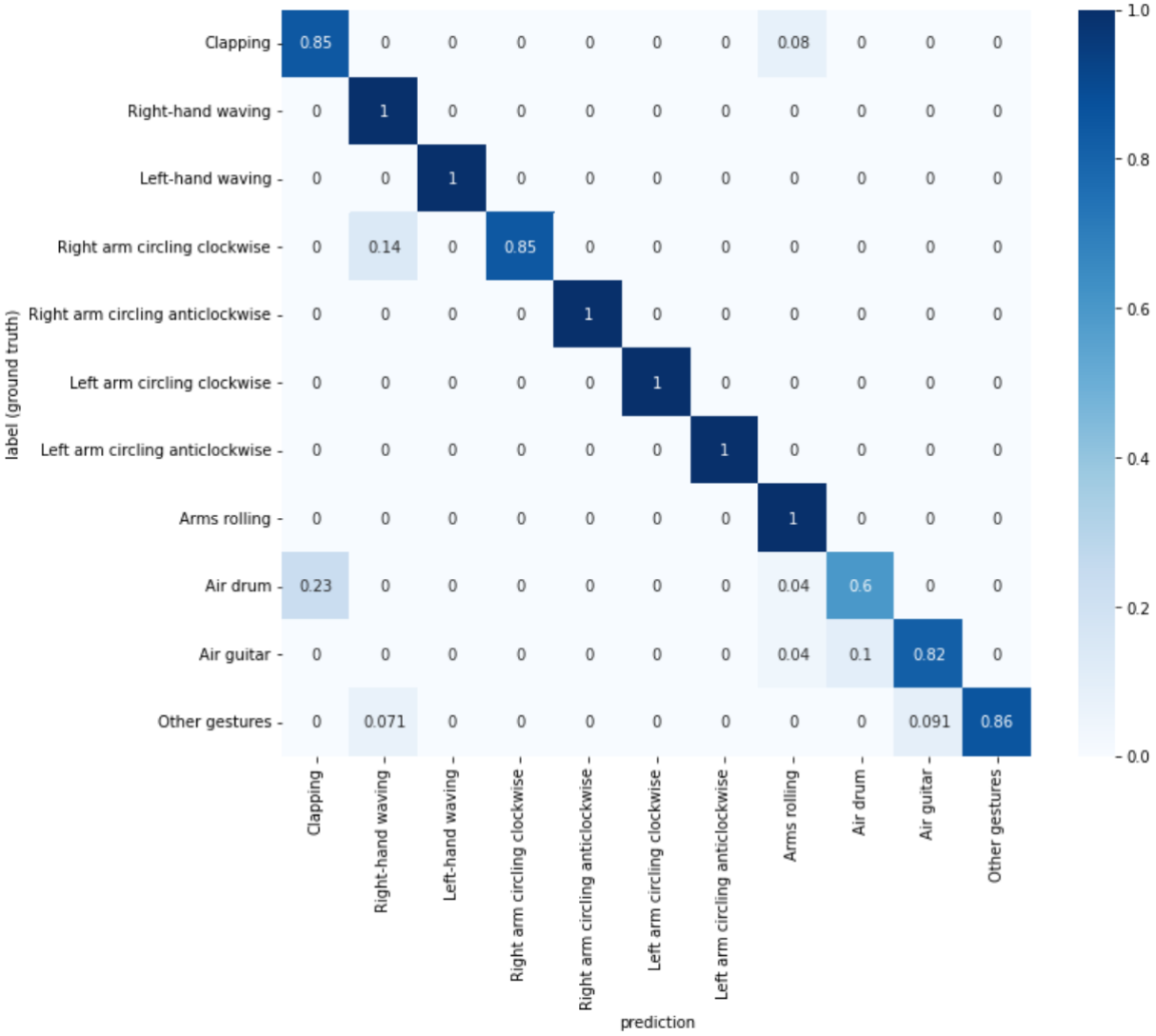

Figure 4.1: Confusion matrix of our model on the final test data set.

CONCLUSION

In this work, we have proposed three different approaches to optimize SNNs and particularly handle the dead neuron issue, all of which have been significantly proved for their performance and give very competitive results. From this work, we have concluded that SNNs have a high potential and can possibly soon outperform conventional ANNs, but this depends on community engagement to push the limits of SNNs.

4.4) Constraints

The progress of this work was limited by a number of limitations, such as:

- Since SNNs are still in the early stage of development, there is no uniform or conventional library or framework to use. Many papers (and their GitHub repository) use different libraries and some create it from scratch, thus reproducing the work in order to optimize it, is quite complicated.
- The time dimension and the sequential behavior of the network make the convergence time very long.
- Another issue is hardware resources, where the offer of Colab Pro + is very limited compared to the simulation demands, for instance, it was hard to maintain a batch of 128 for DVS128 Gesture where it risks crashing at any moment during the simulation, also the Colab do not offer multiple simultaneous simulation of different code, so we were limited to a single simulation at moment (2 in maximum), yet the Colab is the cheapest cloud service, other services such as Google access to GPUs are very expensive (minimum of 0.11 $ per minute).
- Furthermore, the Colab service requires stable internet connection, which is hard to ensure due to the national events, such as Baccalauréat exams and Oran Mediterranean games.

4.5) Future directions

We have wanted to provide more propositions because it was not possible to give time, so we aim to work on:

- Perform an advanced hyperparameter optimization to find those that suit well models. The window length, slope, batch size, learning rate, and many other parameters have not been tuned. Thus, poor values of these hyperparameters can lead to poor performance.
- Finalize the evaluation and test the methods on other larger datasets.
- Test our new approach in the context of object detection.

Appendix A: Basics of Neural Networks

Origin

Among the scientific community, it is often accepted that the human brain is the main source of inspiration and will probably remain so for a long time when it comes to efficiency and the ability to perform several complex tasks. As result, scientist tried at initial stage to mimic biological nervous systems operate using neural networks, these are able to be to learn and adapt its behavior in order to achieve the targeted result. It is about a set of interconnected computational nodes, called artificial neurons, structured in layers and route information flow via link, called synapses, just like the biological counterparts.

Artificial Neuron

As mentioned above, artificial neurons are elementary units, they can be computed with a very simple structure and function, and yet their concatenation can create very complex structures. Intuitively, each node receive information from the previous layer neurons, processing through an activation function and broadcast the produced result to neurons in the next layer. The figure fig. 2 shows an illustration about neuron information processing.

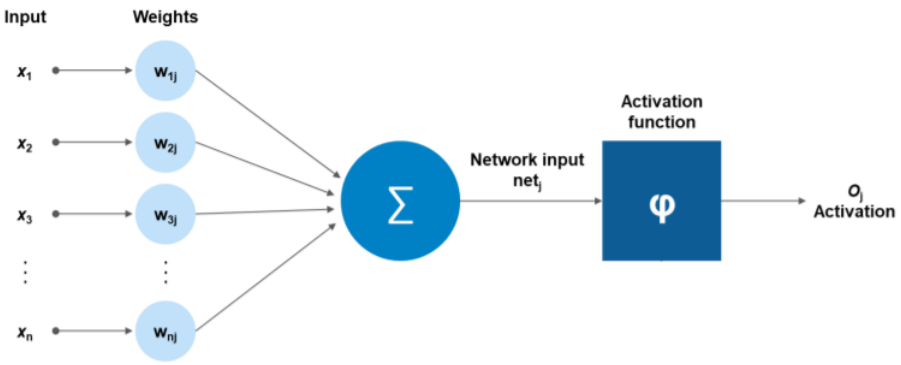

Figure 2: Artificial neuron.

Mathematically, the output of a neuron is represented by the following function:

$$y_j = f\left(\sum_i^N w_i j x_i + b_j\right) = f\left(W^T \times X + B\right)$$

Where:

X / $\mathbf{x_i}$: Inputs from neurons in the previous layer.

W / $\mathbf{w_{ij}}$: Synapse weights connecting input i, with neuron j

B / $\mathbf{b_j}$: Bias of the neuron j.

f: Activation function of the neuron i.

Activation function

Activation function, is almost the most important part of an artificial neuron, it is applied to the weighted sum of the input to introduce a non-linearity on it. To be fully functioning, it has to satisfy some conditions, including the following:

- Nonlinearity: It is very important to enable the model to achieve very complex structures.
- Differentiable: The derivative has to be continuous, in order to train using backpropagation.
- Unsaturation: The derivative has to be neither null nor close to zero. Otherwise, there is the risk that the gradient tends to zero and cannot update the parameters, so the model will never learn.
- Simple calculation: It is not a necessary condition, but a preferred one, it allows simplifying the calculations and gain in training time.
- Monotonic: It is not a necessary condition, but a preferred one, it enables the single layer to be almost guaranteed to be convex; if layers do so then the loss function is convex, and training would be easier.

There are several types of activation function that satisfy these conditions; each can be used in some specific applications. Although we only list those we need, here is a comparison in the following table 16.

	Function	**Propreties**
Step	$f(x) = \begin{cases} 1 & \text{si } 0 < x \\ 0 & \text{sinon.} \end{cases}$	Old-fashioned threshold based function, since its derivative is null, it is unable to perform well in backpropagation and updating weights, and the network would not be able to learn, but it is the most biological one.
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	Behave very similarly as the step function and maps the output between 0 and 1. However, it is derivable which make training more accurate.
ReLU	$f(x) = Max(0,x)$	Rectified Linear Unit (ReLU) is a piece-wise linear function, very popular activation function for its simplicity, it has a constant non saturating gradient that allow a quick calculation of backpropagation.
Softmax	$f(x) = \frac{e^{z_i}}{\sum_j e^{z_i}}$	Usually used at the final layer, it provides the output in probabilistic form for exclusive classification, where the sum of the output is equal to 1.

Table 16: Table of comparison between popular activation functions.

Training neural network

A supervised training process aims to find the optimum parameters that enable the best possible performance, which means being able to predict with the minimum possible error.

Algorithm 1 TRAINING PROCESS OF A NEURAL NETWORK.

Initialization: Assigning random values to parameters, or following a certain distribution, initialization method is important since it controls the convergence time and so prevent convergence issues such as vanishing and exploding gradients.

for *each Epoch* **do**

  for *each Batch* **do**

    - Forwardpass: The batch inputs are passed in the network, and the output layer estimate the value or target class (depend on the problem is regression or classification).

    - Loss estimation: Using the predicted labels or values, the error is calculated using a cost function.

    - Backward pass: It backpropagates the loss function to each layer using backpropagation algorithm, then at each layer we calculated the updated parameters of the layer using gradient descent algorithm.

  end

end

When completing a single batch, an iteration is achieved. When the training is done with all batches, an epoch is completed. The process can be repeated over for several epochs, until targeted performances are achieved.

Cost function

Also known as loss function or objective function, it is an estimation of the error between the prediction of the network and the target estimation. The learning process aims to minimized to reach its global minima, this is why the objective function should be at least convex with respect to its input (but not necessarily parameters). The choice of an objective function is a hyperparameter that depends on the type of problem.

Mean Scare Error

It is used in regression problems, where the network tries to estimate a real value.

$$\mathfrak{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Where:

$\mathfrak{L}$: The loss.

y_i : The targeted output.

$\hat{y}_i$: The estimated output.

N : Number of inputs in the batch.

Cross Entropy

It is used in classification problems, where the network tries to predict the class. A class is represented by a percentage, so the output values are distributions between 0 and 1, and cross entropy is linked to the divergence measurement between the target distribution and the estimated one. But there are two types of classification:

Multi-class classification: The problem is to classify the input among multiple classes C, where an input can only have a single class. In this case, the output should be in the form of percentages and the sum of outputs is 1, and the network attributes the highest percentage to the estimated class. Converting the output values into percentages requires a SoftMax layer. For an output vector $o = [o_1, o_2, , o_C]$ a SoftMax layer is defined a follow:

$$y_i = \sigma(o_i) = \frac{e^{o_i}}{\sum_{j=1}^{C} e^{o_j}}$$

The cross entropy loss for an input is estimated by:

$$\begin{aligned}\mathfrak{L}_i &= -\sum_{j=1}^{C} y_i log(\hat{y}_i) \\ &= -1 \times log(\hat{y}_t) - 0 \times log(\hat{y_{nt}}) - ... - 0 \times log(\hat{y_{nt}}) \\ &= -log(\hat{y}_t)\end{aligned}$$

The total loss for a batch of N input:

$$\mathfrak{L} = \frac{1}{N} \sum_{j=1}^{N} \mathfrak{L}_i$$

Where:

y_t : The target class.

y_{nt} : A non-target class.

C : The number of classes.

N : Number of inputs in the batch.

Multi-label classification: The problem is to classify the input among multiple classes C, where an input can be included in more than one class. In this case, the output should be in the form of independent percentages and the sum of outputs is not necessarily 1. Converting the output values into independent percentages requires a Sigmoid layer. For an output vector $o = [o_1, o_2, , o_C]$ a Sigmoid layer is defined a follow:

$$y_i = \sigma(o_i) = \frac{e^{o_i}}{1 + e^{o_j}}$$

The cross entropy loss for an input is estimated by:

$$\begin{aligned}\mathfrak{L}_i &= -\sum_{j=1}^{C} y_i log(\hat{y}_i) \\ &= -1 \times log(\hat{y_{t_1}}) - 1 \times log(\hat{y_{t_2}}) - 1 \times log(\hat{y_{t_3}}) - 0 \times log(\hat{y_n}t) - ... - 0 \times log(\hat{y_n}t) \\ &= -\sum_{j=1}^{T} log(\hat{y_{tj}})\end{aligned}$$

The total loss for a batch of N input:

$$\mathfrak{L} = \frac{1}{N}\sum_{j=1}^{N} \mathfrak{L}_i$$

Where:

y_{t_i} : Targeted class i.

y_{nt} : Non targeted class.

C : The number of classes.

N : Number of inputs in the batch.

Gradient Descent Algorithm

First order optimization [a] algorithm, where the gradient [b] estimates the direction of the minima and the acceleration toward it, therefore, it is used to update the parameters to get closer to the minima.

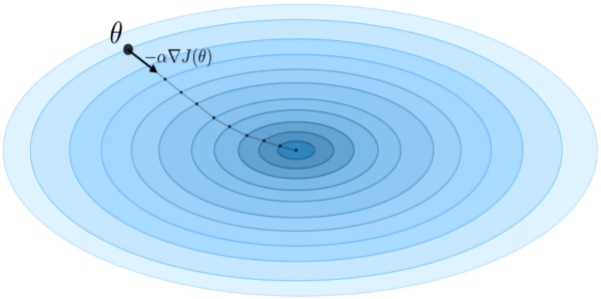

Figure 3: Illustration of the gradient descent algorithm.

For a single parameter, it uses the derivative:

$$p \leftarrow p - \alpha \frac{d\mathfrak{L}}{dp}$$

[a] First order optimization means that the algorithm takes into account only the first derivative.
[b] Generalization of the notion of derivative with multiple parameters.

For a vector of parameters, it uses the gradient:

$$P \leftarrow P - \alpha \nabla \mathfrak{L}$$

Where α is the learning rate. For high values of learning rate, convergence can be quick but risk overshooting the global minima and keeps fluctuating around without reaching it. For low values of the learning rate, the convergence would be very slow or even stuck if the learning rate is decreasing during training. An appropriate choice of learning rate is an important hyperparameter to consider.

BackPropagation Algorithm

It is an algorithm that calculates the gradient with respect to the parameters of each layer using the chain derivation rule. The intuition behind solving the credit assignment problem by estimating the contribution of each parameter to the total loss of the network.

$$\Delta W \; \alpha \; \frac{d\mathfrak{L}}{dW}$$

$$\frac{d\mathfrak{L}}{dW} = \frac{d\mathfrak{L}}{dY} \times \frac{dY}{dW}$$

$$Y(X) = f^{l}\left(W^{L} \times f^{l-1}\left(W^{l-1} \times f^{l-2}\left(W^{l-2} \ldots f^{1}\left(W^{1} \times X\right)\right)\right)\right)$$

$$\frac{dY}{dW} = \frac{dY}{df^{L}} \times \frac{df^{L}}{df^{L-1}} \times \ldots \times \frac{df}{dW}$$

Optimization of the learning process

In theory, the combination of gradient descent and backpropagation works pretty well. However, in practice, there are other parameters to consider during training, such as avoiding local minima and saddle points and reducing training time. Many optimizations can be used to overcome these problems.

Optimizing the convergence time

Vanilla Gradient Descent: It uses the complete training set at once to update parameters, calculates the total loss of all data without batching, then uses gradient descent to optimize, then the process is repeated for another epoch, until it reaches an acceptable performance. It is very useful in updating the parameters, it allows for smooth and soft convergence, but slows the convergence process, and it is inefficient in the case of large datasets.

Stochastic Gradient Descent: Unlike the previous method , updates the parameters after each input. Although the input comes from the same distribution, there is always a variance among those inputs, which affects the training process and makes it more sensitive to these frequent variations that cause more fluctuations; this is the most important drawback

of the algorithm. However, the advantage is a faster training process and is more likely to bypass local minimums.

Mini-Batch gradient descent: The middle-ground between both previous approaches, where it subsets the training set into mini-batches, where it updates after each mini-batch. The length of the minibatch is a hyperparameter that controls the convergence time and fluctuations.

Optimizer

The convergence process can be stuck at local minima and saddle points where the gradient is null, and thus no update occurs and causes him to be stuck in this state. As a result, optimizers consider the history of the gradient to update the hyperparameters to fit the convergence process.

Momentum: It uses the average gradient of the last N steps when updating the gradient, instead of the immediate gradient. The pro of the method is that to reduce the fluctuations by focusing progress on a smooth pathway, it is useful to reduce convergence time since it does not need a higher learning rate.

RMSprop: Alternatively, this method uses exponential average gradients to update the learning rate, the learning rate increases for small gradients and vice versa. It enables reducing fluctuations and smoothing the convergence.

ADAM: It combines both previous techniques, where the learning rate is set as adaptive and the average gradient is included in the parameter update, thus combining both advantages, which explains its popularity, where many publications [52] confirmed that it outperforms the other optimizers.

Underfit and Overfit

During the training process, there are two important parameters that control the quality of the model, bias and variance. The plot fig. 4 shows the progress of bias and variance on learning curves during training.

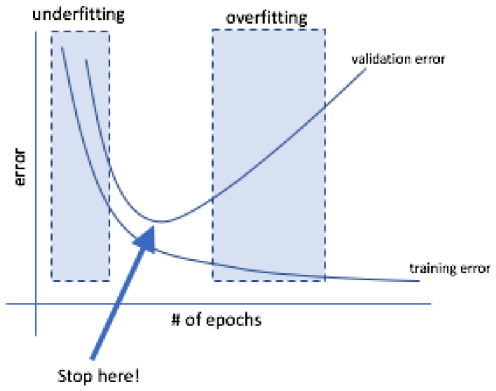

Figure 4: Learning curves of a model during the learning process.

Underfit

It is the inability of the network to learn the training set properties nor the correlation between inputs, it is mainly due to the network architecture that is so simple compared to the required task. Subsequently, the loss is very high, and the training process cannot reduce it; in this case, the network has a high bias between predicted and targeted outputs.

In the first epoch of training, the network still not performing well on train set, in this case it is also an underfit but not related to the architecture, the loss would be reduced over epochs. The figure fig. 5 illustrates the possible learning curves affected by an underfit.

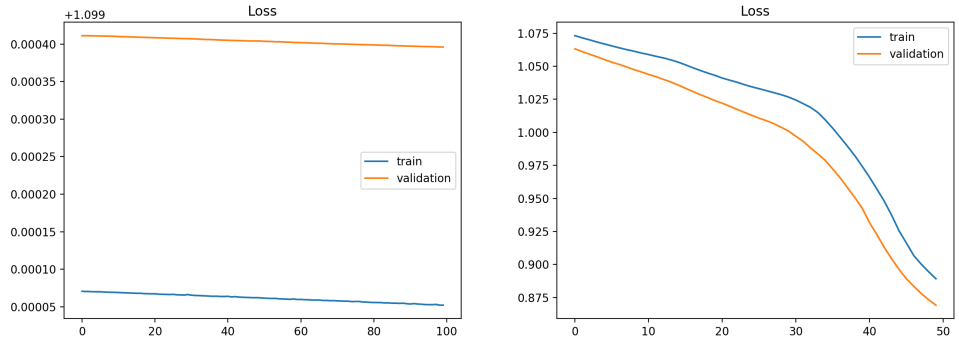

Figure 5: Learning curves of an underfitting model.

Overfit

An input contains information and noise, the noise is random and contributes to the variance of the data set, since it cannot be reduced, the network should be able to be resilient again. Thus, the training process aims to ensure that the network learns only useful information and ignores noise. However, if it captures the noise too, it would not be able to generalize to unseen input. This phenomenon is overfitting, where it has optimum performance on the train set but cannot reproduce those performances on the validation or test set since they are different and unseen. The network variance is high enough to be unable to generalize. The figure fig. 6 illustrates a learning curves affected by an overfit.

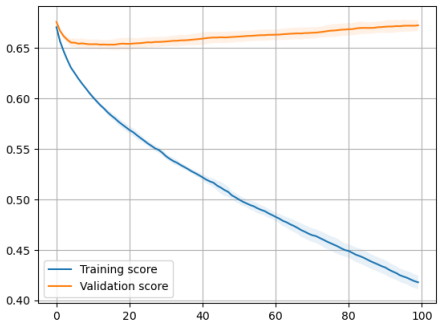

Figure 6: Learning curve of an overfitting model.

NOTES

The curves of the unrepresentative data set (small data set) can be misleading for some time and give the illusion of overfitting. A distinguish mark primarily pertains to the gap between the training loss and the validation loss. The figure fig. 7 indicates the behavior of a model trained with an unrepresentative data set.

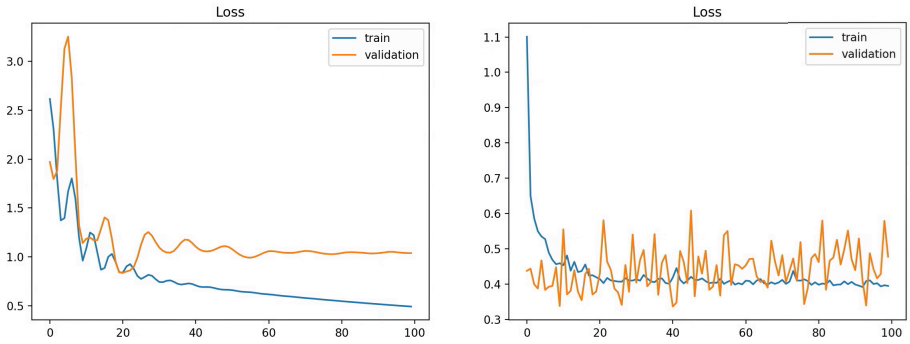

Figure 7: Learning curves of unrepresentative Train / Validation data set respectively.

Regularization

Those are methods used during the training process to prevent the network from overfitting and help the network to generalize more and create a certain balance between bias and variance.

Dropout

The main cause of overfitting is that the parameter distribution is very divergent, where the network relies mainly on some features and ignores others. This high dependency can be reduced by setting randomly (using Bernoulli probability) and momentary some neurons to zero (dropped out) along with all its incoming and outgoing connections. Thus, this unit is not considered during a particular forward and backward pass, at each iteration different neurons are targeted, so the network changes each iteration. This technique helps the network to reduce the dependency to those features and spread parameter distribution, only then the network is less prone to overfit. The figure fig. 8 illustates the principle.

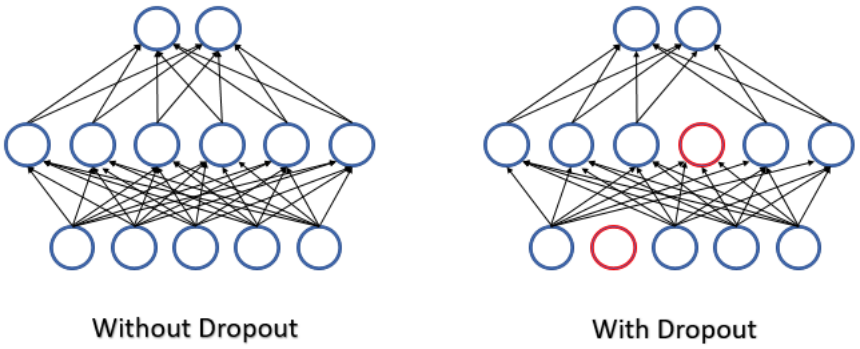

Figure 8: Illustrated example of dropout

NOTES

- In the case of multichannel inputs, very common among Convolutional Neural Networks (CNNs) output, the dropout cannot be random, since the features of the same channel are correlated, thus implicating the use of channel dropout [63].
- In the case of RNNs, dropout must be a fixed mask for each input, not be variable during different sequence of the input, otherwise it would neither prevent nor help the networks against overfitting. This technique is also known as spatial dropout [63].

L1 / L2 regularization

Similarly, the L1 / L2 regularization attempts to spread the parameter distribution and avoid the network that relies only on some features. The intuition behind this method is to add the scaled sum of parameters to the loss function to minimize it.

Early stopping

Training a model passes through three stages as mentioned earlier, yet it is not possible to identify the right number of epochs that guarantee the right fit of the network. Therefore, early stopping can stop the training process before completing the whole number of epochs (which is the minimum validation loss), if the generalization loss [c] increases or stagnates for a long time.

Data augmentation

In contrast to the previous techniques, where overfitting is due to large architecture, Data augmentation handles another cause of overfitting related to the training dataset. Regardless of the split proportions, when the training set is not large enough, the input has relatively small variance, which drives the network to learn noise when there is no added information, thus overfitting. This approach artificially extends the dataset by creating new data from those already existing; the approach assumes that more information can be extracted from the original dataset through augmentations. The creation can be done using, warping, or oversampling. Warping encompasses augmentations such as geometric and color transformations, random erasing, and adversarial training. Oversampling creates synthetic instances by merging images or using . The choice of transformation is largely related to the data set and the problem at hand.

It should be noted that oversampling should be used with caution; if the dataset lakes significantly to variance, it can lead to creating more similar data and lead to having more overfitting.

[c] Generalization loss is the difference between training loss and validation loss.

Normalization

Normalization is one of the techniques that contributes to the success of ANNs. Although it is not used primarily for regularization, but it has been shown to have a slight efficiency against it [32], normalization reduces the internal covariate shift and is effective in stabilizing the hidden state of RNNs, leading to a smooth landscape.

During the training phase, if we estimate the mean and standard deviation of the input, then we use those values in the evaluation and test phase. However, the mean and deviation are estimated in batches, which introduce some error; this error contributes to the generalizability of the network [55], and this is how it fights against overfitting.

Although batch normalization is the most widely used technique, other techniques can be used, such as layer normalization, group normalization, and instance normalization, each outperforming others under special conditions. The figure fig. 9 shows different normalization techniques.

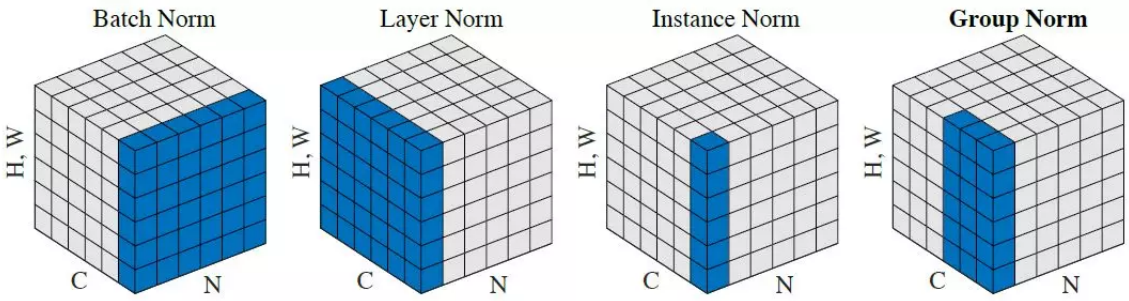

Figure 9: Types of normalization methods.

Reducing the architecture

When all previous techniques are not sufficiently effective in reducing overfitting, it may be the problem of a large architecture, where it has more parameters compared to the features; that is, the network is more complex than the required task.

Benchmarking Neural Networks

Cross validation

It is a statistical method used to select the best model for a specific problem, where it estimates the surrogate performance of each possible model and uses those performances to select the best model for training. Many methods exist, but the most popular are:

K-Fold

It split the data set into K subsets, where $K-1$ subsets are used for training while the rest are used for testing. The subset test can be one of K subsets; each scenario constitutes a folder, so there are K folders of the same data but with different splits. We train the model

on each folder and estimate the performance; the final performance is the average of all the performance of all the folders. Using this method, all samples contribute to training for $K-1$ time and contribute to testing once; thus, all datasets are used for training, which is very useful in the case of an unrepresentative dataset, but the drawback is that it is very computationally expensive and slow.

Hold Out Method

Or Train/Test approach, it is a particular simple case of K-Fold where $K = 2$. Simple and efficient method, but not adapted to a shallow data set, where it could be possible that the training data set is not representative of the entire data sample.

Train/Validation/Test

Train/Validation/Test split to evaluate the model, a particular case of K-Fold Cross Validation. As stated before, it splits the dataset randomly into three sets, one used for the training and reducing the bias, the second used for validation of the model and reducing the variance, train and validation sets are used for tuning parameters, and the test set is for the final performance evaluation. This method is useful not only for model choice but also for model building (training).

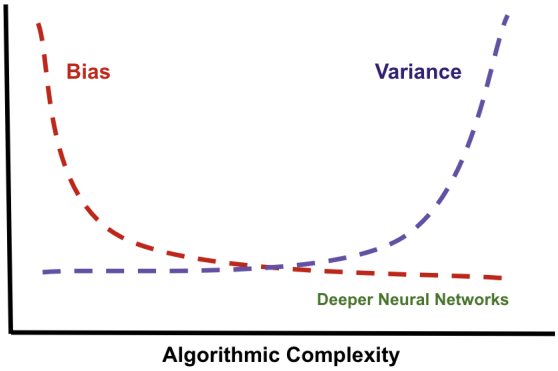

Figure 10: Balance bias variance [6].

Evaluation metrics

Performance metric choice is largely depending on the problem at hand. For classification problems, there is a set of possible metrics:

Accuracy

Percentage of correct estimation over the total number of estimations.

$$\text{Accuracy} = \frac{T}{T+F}$$

Accuracy can be a misleading metric in the case of an unbalanced dataset, where it may correctly class the most apparent class and miss the underrepresented classes, although this side would not be highlighted by accuracy measurement. Other metrics are necessary to better analyze the performance of the network.

Precision and Recall

Used in binary classification mainly for biomedical applications, where the precision of a class is the correct classification of this class over the total classification of the class, and the recall of a class estimates the correct classification over the total number of samples of the real class.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision helps identify how accurate the class prediction is, while recall indicates how accurate the class sample is classified correctly. However, these metrics are available only for binary classification.

F1-Score

It includes both precision and recall in the same formula, using a harmonic mean:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Confusion Matrix

It is a generalization of the notion of precision, recall, sensitivity, and specificity for multi-class classifications.

Other types of Neural Networks

CNN

CNNs are a class of ANNs, where it is used for processing data that have a grid-like topology, where it uses the convolution operation and sub-sampling to extract meaningful features. It has demonstrated high performance on complex AI and ML tasks, such as processing time series (1D), images (2D), videos (3D). The network is a cascade of convolution layer and pooling layer as shown in the figure fig. 11

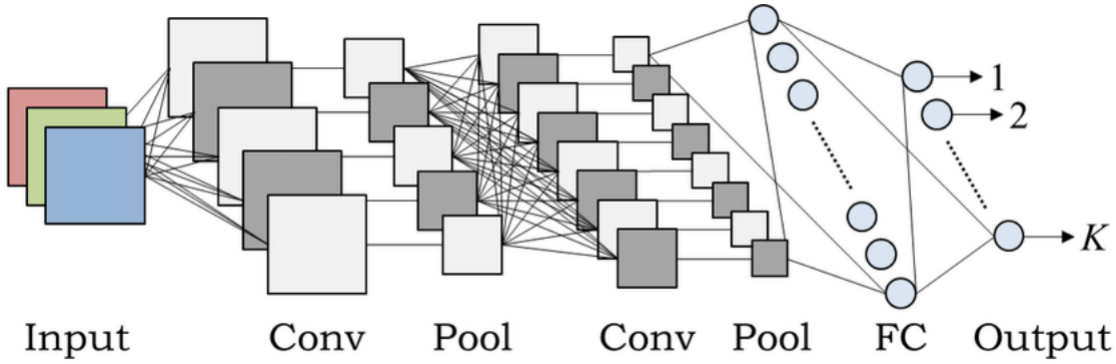

Figure 11: CNN architecture.

Convolution layer

The convolution layer is a fundamental building block that uses a predefined number of filters (also known as kernels) of a predefined size to extract features using mathematical convolution. The convolution 2D is a weighted sum of input pixels that are spatially located within the filter size, when the filter passes sequentially through the image and generates for each grid a value, when accumulating spatially those values, it constitutes the output feature map, the filter weights are parameters adjusted during training using backpropagation. The number and size of filters are hyperparameters of this layer, and it also has other hyperparameters such as padding and stride.

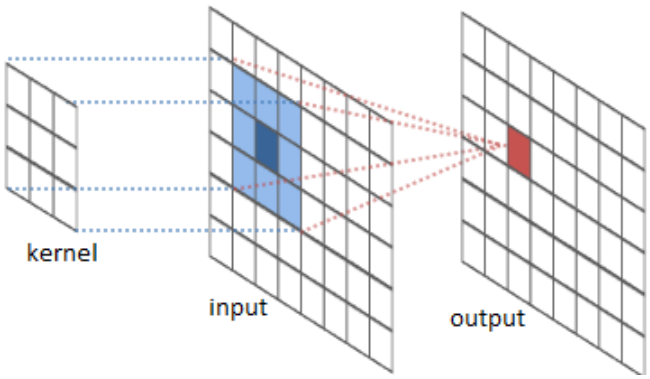

Figure 12: Illustration of the convolution operation.

Pooling layer

It is a subsampling layer that downsampling the feature map produced by the convolutional layer, to preserve the most canonical features. The pooling operator uses another filter that can take the average of the area or its maximum, the first technique is average pooling and the second is max pooling.

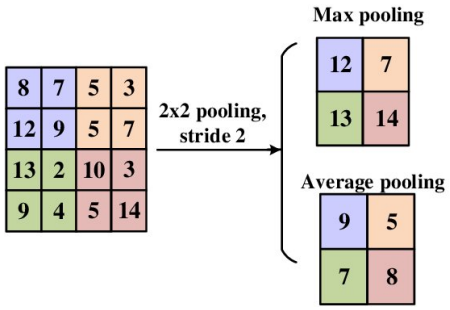

Figure 13: Example of Max Pooling layer.

The choice of max or average pooling depend on the input and task. For example, in the case of black background max pooling is more effective than average pooling, for white background it is preferred to use min pooling, meanwhile for random background average pooling introduce less noise.

RNN

Recurrent Neural Networks (RNNs) are a particular type of ANNs, where their architecture admits beside feedforward connections, inter-neurons and inter-layers connections, those connections are time sensitive unlike the feedforward ones. This architecture enables the network to memorize previous outputs, which allows processing sequential data, such as time series, images sequences (videos), and all time dependent tensors.

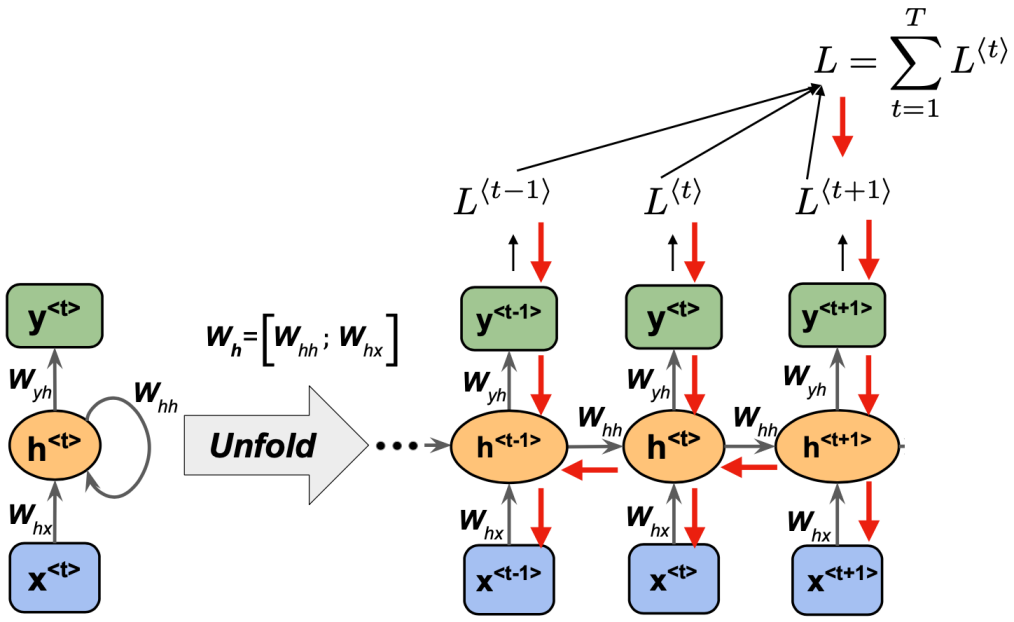

Figure 14: Example of forward and backward pass on unfolded RNNs cell.

RNNs flexibility allows defining multiple possible architectures for different problems. However, training those algorithms require BPTT which consist of unrolling the RNNs computational graph in time, so the recurrent connections are equivalent to forward ones, which allows applying conventional backpropagation on the sum of the loss over time steps.

BIBLIOGRAPHY

[1] Larry F Abbott and Sacha B Nelson. "Synaptic plasticity: taming the beast." In: *Nature neuroscience* 3.11 (2000), pp. 1178–1183.

[2] *AI and Compute*. URL: https://openai.com/blog/ai-and-compute/.

[3] Arnon Amir, 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. "A Low Power, Fully Event-Based Gesture Recognition System." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[4] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. "Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models." In: *CoRR* abs/2007.03051 (2020). arXiv: 2007.03051. URL: https://arxiv.org/abs/2007.03051.

[5] Daniel Auge, Julian Hille, Etienne Mueller, and Alois Knoll. "A survey of encoding techniques for signal processing in spiking neural networks." In: *Neural Processing Letters* 53.6 (2021), pp. 4693–4710.

[6] *Balancing the Regularization Effect of Data Augmentation*. https://towardsdatascience.com/balancing-the-regularization-effect-of-data-augmentation-eb551be48374. Accessed: 2022-06-12.

[7] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. "A solution to the learning dilemma for recurrent networks of spiking neurons." In: *Nature communications* 11.1 (2020), pp. 1–15.

[8] Guo-qiang Bi and Mu-ming Poo. "Synaptic modification by correlated activity: Hebb's postulate revisited." In: *Annual review of neuroscience* 24.1 (2001), pp. 139–166.

[9] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[10] Sander M Bohte. “The evidence for neural information processing with precise spike-times: A survey.” In: *Natural Computing* 3.2 (2004), pp. 195–206.

[11] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. “SpikeProp: backpropagation for networks of spiking neurons.” In: *ESANN*. Vol. 48. Bruges. 2000, pp. 419–424.

[12] Sander M Bohte, Han La Poutré, and Joost N Kok. “Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks.” In: *IEEE Transactions on neural networks* 13.2 (2002), pp. 426–435.

[13] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. “Language Models are Few-Shot Learners.” In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[14] Anthony N Burkitt. “A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input.” In: *Biological cybernetics* 95.1 (2006), pp. 1–19.

[15] Yunhua Chen, Yingchao Mai, Ren Feng, and Jinsheng Xiao. “An adaptive threshold mechanism for accurate and efficient deep spiking convolutional neural networks.” In: *Neurocomputing* 469 (2022), pp. 189–197.

[16] Xiang Cheng, Yunzhe Hao, Jiaming Xu, and Bo Xu. “LISNN: Improving spiking neural networks with lateral interactions for robust object recognition.” In: *IJCAI*. 2020, pp. 1519–1525.

[17] Loc Cordone, Benot Miramond, and Sonia Ferrante. “Learning from event cameras with sparse spiking convolutional neural networks.” In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[18] Shirin Dora and Nikola Kasabov. “Spiking Neural Networks for Computational Intelligence: An Overview.” In: *Big Data and Cognitive Computing* 5.4 (2021), p. 67.

[19] Jason K Eshraghian and Wei D Lu. “The fine line between dead neurons and sparsity in binarized spiking neural networks.” In: *arXiv preprint arXiv:2201.11915* (2022).

[20] Jason Kamran Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. “Training Spiking Neural Networks Using Lessons From Deep Learning.” In: *CoRR* abs/2109.12894 (2021). arXiv: 2109.12894. URL: https://arxiv.org/abs/2109.12894.

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

[22] Joseph J Feher. *Quantitative human physiology: an introduction*. Academic press, 2017.

[23] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. "Event-based vision: A survey." In: *IEEE transactions on pattern analysis and machine intelligence* 44.1 (2020), pp. 154–180.

[24] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

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

[26] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[27] Julian Göltz, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Dominik Dold, Laura Kriener, Ákos Ferenc Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A. Petrovici. "Fast and deep neuromorphic learning with time-to-first-spike coding." In: *CoRR* abs/1912.11443 (2019). arXiv: 1912.11443. URL: http://arxiv.org/abs/1912.11443.

[28] Weihua He, Yujie Wu, Lei Deng, Guoqi Li, Haoyu Wang, Yang Tian, Wei Ding, Wenhui Wang, and Yuan Xie. "Comparing SNNs and RNNs on Neuromorphic Vision Datasets: Similarities and Differences." In: *CoRR* abs/2005.02183 (2020). arXiv: 2005.02183. URL: https://arxiv.org/abs/2005.02183.

[29] Allan L Hodgkin and Andrew F Huxley. "Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo." In: *The Journal of physiology* 116.4 (1952), p. 449.

[30] Shin-ichi Ikegawa, Ryuji Saiin, Yoshihide Sawada, and Naotake Natori. "Rethinking the role of normalization and residual blocks for spiking neural networks." In: *Sensors* 22.8 (2022), p. 2876.

[31] *Introduction to Theoretical Neuroscience*. URL: https://warwick.ac.uk/fac/sci/systemsbiology/staff/richardson/teaching/ma4g4/ITN_LN6.pdf.

[32] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[33] Hyeryung Jang, Nicolas Skatchkovsky, and Osvaldo Simeone. "Spiking Neural Networks - Part I: Detecting Spatial Patterns." In: *CoRR* abs/2010.14208 (2020). arXiv: 2010.14208. URL: https://arxiv.org/abs/2010.14208.

[34] James M. Joyce. “Kullback-Leibler Divergence.” In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_327. URL: https://doi.org/10.1007/978-3-642-04898-2_327.

[35] Jacques Kaiser, Alexander Friedrich, Juan Camilo Vasquez Tieck, Daniel Reichard, Arne Roennau, Emre Neftci, and Rüdiger Dillmann. “Embodied Event-Driven Random Backpropagation.” In: *CoRR* abs/1904.04805 (2019). arXiv: 1904.04805. URL: http://arxiv.org/abs/1904.04805.

[36] Michael J Kearns. *The computational complexity of machine learning*. MIT press, 1990.

[37] Saeed Reza Kheradpisheh and Timothée Masquelier. “Temporal backpropagation for spiking neural networks with one spike per neuron.” In: *International Journal of Neural Systems* 30.06 (2020), p. 2050027.

[38] Youngeun Kim and Priyadarshini Panda. “Visual Explanations from Spiking Neural Networks using Interspike Intervals.” In: *CoRR* abs/2103.14441 (2021). arXiv: 2103.14441. URL: https://arxiv.org/abs/2103.14441.

[39] Alexander Kugele, Thomas Pfeil, Michael Pfeiffer, and Elisabetta Chicca. “Efficient processing of spatio-temporal data streams with spiking neural networks.” In: *Frontiers in Neuroscience* 14 (2020), p. 439.

[40] Louis Lapicque. “Louis Lapicque.” In: *J. physiol* 9 (1907), pp. 620–635.

[41] Eimantas Ledinauskas, Julius Ruseckas, Alfonsas Jursenas, and Giedrius Burachas. “Training Deep Spiking Neural Networks.” In: *CoRR* abs/2006.04436 (2020). arXiv: 2006.04436. URL: https://arxiv.org/abs/2006.04436.

[42] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. “Enabling spike-based backpropagation for training deep neural network architectures.” In: *Frontiers in neuroscience* (2020), p. 119.

[43] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. “Training Deep Spiking Neural Networks Using Backpropagation.” In: *Frontiers in Neuroscience* 10 (2016). ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508. URL: https://www.frontiersin.org/article/10.3389/fnins.2016.00508.

[44] Yuhang Li, Youngeun Kim, Hyoungseob Park, Tamar Geller, and Priyadarshini Panda. *Neuromorphic Data Augmentation for Training Spiking Neural Networks*. 2022. DOI: 10.48550/ARXIV.2203.06145. URL: https://arxiv.org/abs/2203.06145.

[45] Sen Lu and Abhronil Sengupta. “Exploring the Connection Between Binary and Spiking Neural Networks.” In: *CoRR* abs/2002.10064 (2020). arXiv: 2002.10064. URL: https://arxiv.org/abs/2002.10064.

[46] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. “Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs.” In: *Science* 275.5297 (1997), pp. 213–215.

[47] Marek Miskowicz. “Send-on-delta concept: An event-based data reporting strategy.” In: *sensors* 6.1 (2006), pp. 49–63.

[48] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. “First-spike-based visual categorization using reward-modulated STDP.” In: *IEEE transactions on neural networks and learning systems* 29.12 (2018), pp. 6178–6190.

[49] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. “Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks.” In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63. DOI: 10.1109/MSP.2019.2931595.

[50] MISHA Perouansky, ROBERT A Pearce, and HUGH C Hemmings Jr. “Inhaled anesthetics: mechanisms of action.” In: *Miller’s anesthesia* 1 (2010), pp. 515–538.

[51] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. “Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation.” In: *arXiv preprint arXiv:2005.01807* (2020).

[52] Sebastian Ruder. “An overview of gradient descent optimization algorithms.” In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[53] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. “Going Deeper in Spiking Neural Networks: VGG and Residual Architectures.” In: *Frontiers in Neuroscience* 13 (2019). ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00095. URL: https://www.frontiersin.org/article/10.3389/fnins.2019.00095.

[54] Ahmed Shaban, Sai Sukruth Bezugam, and Manan Suri. “An adaptive threshold neuron for recurrent spiking neural networks with nanodevice hardware implementation.” In: *Nature Communications* 12.1 (2021), pp. 1–11.

[55] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning.” In: *Journal of big data* 6.1 (2019), pp. 1–48.

[56] Sumit Bam Shrestha and Garrick Orchard. “SLAYER: Spike Layer Error Reassignment in Time.” In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/82f2b308c3b01637c607ce05f52a2fed-Paper.pdf.

[57] Per Jesper Sjöström, Gina G Turrigiano, and Sacha B Nelson. “Rate, timing, and cooperativity jointly determine cortical synaptic plasticity.” In: *Neuron* 32.6 (2001), pp. 1149–1164.

[58] Nicolas Skatchkovsky, Hyeryung Jang, and Osvaldo Simeone. “Spiking Neural Networks - Part II: Detecting Spatio-Temporal Patterns.” In: *CoRR* abs/2010.14217 (2020). arXiv: 2010.14217. URL: https://arxiv.org/abs/2010.14217.

[59] Sen Song, Kenneth D Miller, and Larry F Abbott. “Competitive Hebbian learning through spike-timing-dependent synaptic plasticity.” In: *Nature neuroscience* 3.9 (2000), pp. 919–926.

[60] Cheston Tan, Stephane Lallee, and Garrick Orchard. “Benchmarking neuromorphic vision: lessons learnt from computer vision.” In: *Frontiers in neuroscience* 9 (2015), p. 374.

[61] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony S. Maida. “Deep Learning in Spiking Neural Networks.” In: *CoRR* abs/1804.08150 (2018). arXiv: 1804.08150. URL: http://arxiv.org/abs/1804.08150.

[62] *The Future of AI is Decentralized*. URL: https://towardsdatascience.com/the-future-of-ai-is-decentralized-848d4931a29a.

[63] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. “Efficient Object Localization Using Convolutional Networks.” In: *CoRR* abs/1411.4280 (2014). arXiv: 1411.4280. URL: http://arxiv.org/abs/1411.4280.

[64] Jilles Vreeken. “Spiking neural networks, an introduction.” In: 2003.

[65] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. “Supervised learning in spiking neural networks: A review of algorithms and evaluations.” In: *Neural Networks* 125 (2020), pp. 258–280. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.02.011. URL: https://www.sciencedirect.com/science/article/pii/S0893608020300563.

[66] Stanisaw Woniak, Angeliki Pantazi, Thomas Bohnstingl, and Evangelos Eleftheriou. “Deep learning incorporating biologically inspired neural dynamics and in-memory computing.” In: *Nature Machine Intelligence* 2.6 (2020), pp. 325–336.

[67] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.” In: *CoRR* abs/1708.07747 (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

[68] Yannan Xing, Gaetano Di Caterina, and John Soraghan. “A new spiking convolutional recurrent neural network (SCRNN) with applications to event-based hand gesture recognition.” In: *Frontiers in neuroscience* 14 (2020), p. 1143.

[69] Bojian Yin, Federico Corradi, and Sander M. Bohté. “Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks.” In: *bioRxiv* (2021). DOI: 10.1101/2021.03.22.436372. eprint: https://www.biorxiv.org/content/early/2021/03/23/2021.03.22.436372.full.pdf. URL: https://www.biorxiv.org/content/early/2021/03/23/2021.03.22.436372.

[70] Qiang Yu, Chenxiang Ma, Shiming Song, Gaoyan Zhang, Jianwu Dang, and Kay Chen Tan. “Constructing accurate and efficient deep spiking neural networks with double-threshold and augmented schemes.” In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[71] Friedemann Zenke and Tim P Vogels. “The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks.” In: *Neural Computation* 33.4 (2021), pp. 899–925.

[72] Wenrui Zhang and Peng Li. “Spike-train level backpropagation for training deep recurrent spiking neural networks.” In: *Advances in neural information processing systems* 32 (2019).

[73] Wenrui Zhang and Peng Li. “Temporal spike sequence learning via backpropagation for deep spiking neural networks.” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12022–12033.

[74] “CHAPTER 19 - NEAR REAL-TIME ROBUST FACE AND FACIAL-FEATURE DETECTION WITH INFORMATION-BASED MAXIMUM DISCRIMINATION.” In: *Face Processing*. Ed. by Wenyi Zhao and Rama Chellappa. Burlington: Academic Press, 2006, pp. 619–646. ISBN: 978-0-12-088452-0. DOI: https://doi.org/10.1016/B978-012088452-0/50020-0. URL: https://www.sciencedirect.com/science/article/pii/B9780120884520500200.

[75] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. “Going deeper with directly-trained larger spiking neural networks.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 11062–11070.

[76] Romain Zimmer, Thomas Pellegrini, Srisht Fateh Singh, and Timothée Masquelier. “Technical report: supervised training of convolutional spiking neural networks with PyTorch.” In: *CoRR* abs/1911.10124 (2019). arXiv: 1911.10124. URL: http://arxiv.org/abs/1911.10124.