# Spiking Neural Networks, classifying Dynamic Vision Sensor Data

Georgios Alexakis,Dimitrios Korakobounis

2021

# Contents

# Chapter 1

# The Biological Brain

The scale of the mammalian brain is immense. Each human brain comprises of about 25 thousand neurons and $10x10^8$ synapses per cubic centimeter [1] in the neocortex.The neocortex is the outer layer of the brain. It is estimated at 20 years of age, the neocortex in total, contains about $25x10^9$ neurons and almost 180 trillion synapses! We will explore some findings from brain research and see how these can help us develop more efficient machine learning methods.

## 1.1 Brain Structural Elements

### 1.1.1 Neurons

This chapter starts by defining the neuron, the most basic unit of the brain [2]. These cells are responsible for processing sensory feedback from the outside world, and transforming and processing electrical signals [3]. Ramón y Cajal´ was the first researcher to make drawings of neurons after observing them under a microscope [4] .One example of these drawings is depicted in Figure.1.1 .For a better understanding of what a neuron is consisted of see Figure.3.1.

Neurons can cluster(neuronal assemblies) together in the brain and can span 1–2 mm in space and last hundreds of milliseconds. As a result, they may be able to link bottom-up, micro-scale events with top-down, macro-scale events [5] .

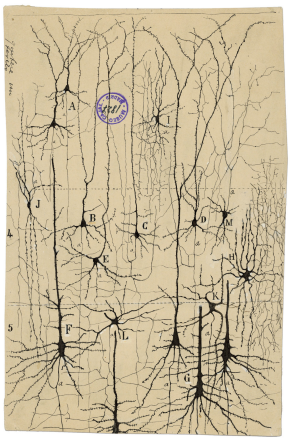


Figure 1.1: The rough surface of dendrites, which leave the cell laterally and upwardly, is what identifies them. The axons are small, straight lines that reach downwards with several branches to the left and right.Ramon y Cajal(1909).

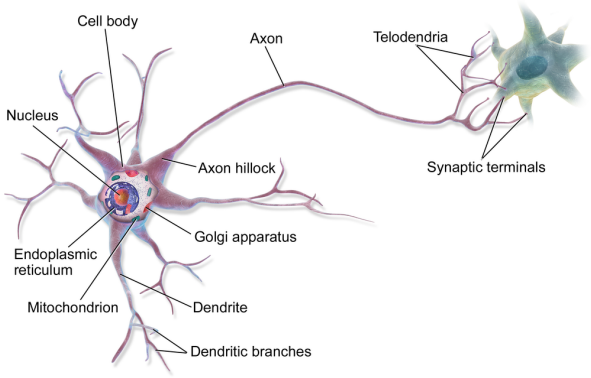



Figure 1.2: Neuron illustration.The dendrites,axons and synaptic terminals are of interest.

### 1.1.2 Synapses

The brain is made up of a vast network of neurons. Neurons communicate with one another through synapses, which are specialized cell junctions. Synapses are required not only for neuronal signaling and computation but also for long-term changes (synaptic plasticity) that underpin information storage, such as learning and memory, in the brain [6]. The contact region between two communicating neurons is defined by two distinct components: the pre-synaptic terminal and the post-synaptic target site, which are separated by a synaptic cleft .

Chemical and electrical synapses have been discovered in research, but we shall concentrate purely on electrical synapses. Electrical synapses cause a pre-synaptic impulse to be quickly converted into an electrical excitatory postsynaptic potential in the post-junctional cell. Activation of voltage-gated ion channels leads to the generation of action potentials if the current transmitted to the post-synaptic cell is sufficient to depolarize the membrane above a certain threshold [7]. The amount of excitation in both cells, however, is not equal. A less depolarized paired partner can be excited by a more depolarized cell, and a more depolarized cell can be inhibited by a less depolarized cell.The synapse can also cause a rectifying behavior [8].Electrical synapses are very intriguing for researchers due to their unique ability of reciprocity as well as their ability to carry sub-threshold potentials allowing for synchronous activity of neurons.

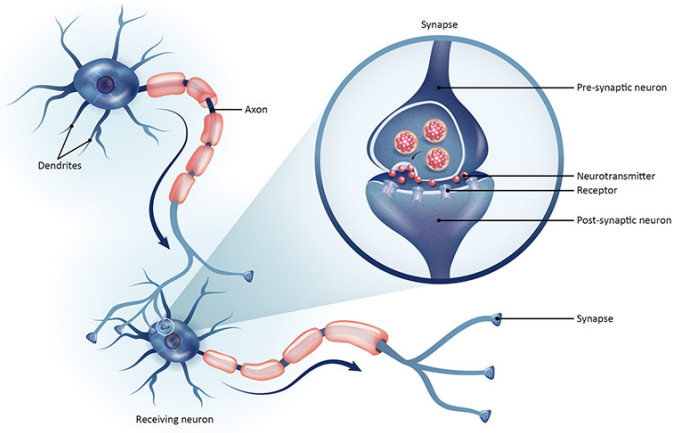



Figure 1.3: Synapse

It has been observed that gap junctions serve an important role in the development of the nervous system . They also produce large functional clusters of coupled neurons, which are usually organized in vertical columns spanning many cortical layers [9] [10] [11] 1.5 .

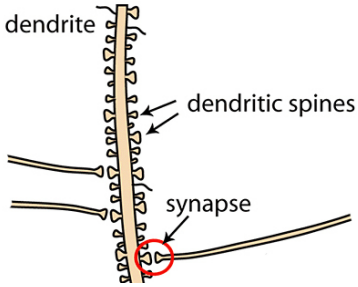



Figure 1.4: Synapse

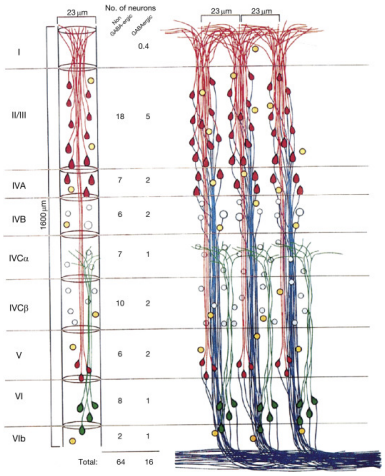



Figure 1.5: Vertical Columns [12]

Since we are interested in how findings from brain research can be applied in computer vision machine learning applications it is necessary to understand how synapses aid in processing vision input in the mammalian brain. Inputs from pre-synaptic neurons are transmitted through tiny protrusions called spines on the post synaptic dendrites(see Fig.1.4) [13].However, where an input is located on the dendritic tree, as well as whether it is triggered by similar stimuli to those that trigger its neighbors, matters, allowing for simultaneously active inputs [14].

For a long time, it was unclear what information each neuron receives from various parts of the visual field, and how this information relates to the visual features encoded by the neuron's spatial receptive field until recent research has provided insight. Inputs representing similar visual features from the same location in visual space were more likely to cluster on neighbouring spines.Higher-order dendritic branches often synapse inputs from visual field regions beyond the postsynaptic neuron's receptive field. When the input's receptive field is spatially displaced along the axis of the postsynaptic neuron's receptive field direction, these putative long-range inputs are more frequent and more likely to share the postsynaptic neuron's preference for oriented edges. As a result, neurons with displaced receptive fields bind preferentially when their receptive fields are co-oriented and co-axially aligned. This synaptic connectivity organization is well suited for amplifying elongated edges and thus serves as a possible "framework" for contour integration(Contour integration refers to the ability of the visual system to bind disjoint local elements into coherent global shapes [15]) and object grouping [16] and provides evidence for the idea that visual space is mapped onto the dendrites in a specific manner.

## 1.2 Information Representation and Processing in the Brain

After reviewing the functional elements of the brain, it's time to look at how information is represented in the cerebral cortex with spike trains of neuronal populations and also how can this research provide a better methodology for machine learning research.

### 1.2.1 Neural representation

A message that uses the rules and structures by which a signal carries information (neural code) to serve a function is called a neural representation. Its content and function make up the representation. The signal that carries a sensory input is the representation's content, while its function is the processing of the input sensory signal (cognitive process) and the cognitive process' outcome.

Organisms produce an internal mirror that correctly reflects their environment [17]. The input signal must then have projections that enable it

to play a role in the organism's activities to adapt to its surroundings. The mechanism of transforming representations is called representation. Computation is a complement to representation. The knowledge transformations that representations serve would be impossible without computational processes. As neuronal representations are projected from one cortical area to another, they are often transformed. Neural circuits represent information as they operate, which they then transform using computational processes [18].

### 1.2.2 Neural Coding

How information is exactly represented and processed in the brain is still an open question to neuroscience but several hypotheses exist. The rate-coding( average number of spikes over some time interval) hypothesis [19] [20] contends that the information is carried by the mean firing rate, while the temporal-coding hypothesis [21] [22] [23] claims that the exact timing of the spikes is what encodes the information carried. The distinction between these two hypotheses is the interval used to count the spikes, meaning that in temporal coding the interval is so small that only one spike is measured. The interval for an experiment that uses rate encoding is decided upon the timescales thought to be important to a given situation, such as how rapidly the stimulus shifts, the integration time of a neural variable, the encoding process, or the relevant behavioral timescale.

A spike train is a complex time-varying signal consisted of multiple spikes produced by the neuron at certain times. The rate-coding hypothesis considers only one number to be important, the mean rate of these spikes. Even though encoding and decoding(the neuronal response is decoded by counting the spikes, and the stimulus is encoded by setting the firing rate proportional to the value of some stimulus parameter) are simple this hypothesis appears to be an oversimplification. In behavioral experiments, response times are often too short to allow for slow temporal averaging [24]. In another experiment on a visual neuron of a fly, the time dependent stimulus was successfully reconstructed from neuron firing times [25]. There is also evidence of precise temporal correlations between pulses of different neurons [26] .

#### Neuron Assemblies

Yoshio Sakurai in his research [27] states that single neurons are inadequate as a basic coding mechanism. As we mentioned earlier the brain contains

an uncountable number of synapses, that means that each neuron receives signals from thousands other neurons. This make a single neuron to have an unstable firing behavior as their membrane potential undergoes large fluctuations.

Additionally, individual single neurons have only quite minimal influence on other neurons and cannot produce a strong enough transmission to trigger spikes in the next neurons when it comes to functional transmissions between neurons. According to theoretical arguments, the brain's number of neurons is insufficient to capture the enormous amount of data that an animal processes over its lifetime. Because combinations and configurations of items produce new items, there is a combinatorial explosion, the number of information items is nearly limitless. Single-neuron representation of such items is also inconvenient for associating and discriminating among information items, indicating the degree of similarity or difference among items, or creating new concepts and ideas from disparate pieces of data. Thus, the single neuron hypothesis seems like unlikely to explain the brain's coding and processing. The ensemble activity of a population of neurons seems more capable of encoding information in the brain. This hypothesis is called ensemble coding.

To give you a more exact definition: A set of neurons forms a functional group if the impulses are coordinated, to the extent that their temporal relationships are arranged, at least probabilistically, in characteristic patterns is what how an older research paper defines a neuron assembly [28] . A neuron does not have to engage in just one functional groups; it can be part of multiple neuronal assemblies at different times. Furthermore, the neurons that make up a functional group do not need to be in direct synaptic contact or in close proximity anatomically. Neuronal assemblies can also form when neurons share extrinsic input activity. Because we'll be focusing on visual processing in this dissertation, let's examine at how neuronal assemblies can play a role in the various stages of visual processing.

Certain processes that take place during a period of postnatal development result in the selective stabilization of connections between neuronal components that often have linked activity, allowing connectivity to be modified based on functional parameters [29]. In the input stage of the striate cortex (primary visual cortex) neuronal assemblies can contribute to optimize the match between the representations of the two eyes. It also helps to construct neuronal representations for frequently recurring feature configurations at a later level of processing by participating in the formation of

selective connections between cortical columns .

The enhanced correlated activity can occur from selective enhancement in synaptic connections among neurons.Such selective enhancement refers to the temporary dynamics of that is sustained during a neural activity, rather than a permanent change in synaptic efficacy between neurons. As a result, the above-mentioned event- and behavior-related dynamic modulation of correlated firing in neurons supports the idea that neurons can rapidly associate into a functional group in order to process the required information while remaining dissociated from concurrently activated competing groups. Finally, let us examine important properties of Ensemble coding :

1. Overlapping set coding of information items. The same neuron is a part of many different assemblies.

2. Sparse coding of information items. Any individual cell assembly contains a small subset of all of the neurons in the cortex. This property makes it interesting for machine learning researchers that want to look into more efficient sparse computations.

3. Dynamic construction and reconstruction. Cell assemblies are temporal sets of neurons interconnected by flexible functional synapses. This enables very flexible behavior something that is almost not existent in current machine learning methods.

4. Dynamic persistence . Activation of a cell assembly will persist for a time via feedback due to the excitatory synapses among the neurons.

5. Dynamic completion. Activation of a large enough subset of a cell assembly results in activation of the complete cell assembly.

The neural ensemble coding hypothesis is an exciting one since it can be confirmed by many research studies in the human brain.Functional overlapping of individual neurons and correlation dynamics among multiple neurons were found in working memory and reference memory. Cell assemblies are composed of task-related single neurons—a possibility for dual coding by cell assemblies and their single neuron. The experimental data for these experiments can be found in [27]. Understanding the meaningful results behind these experiments might enable for more efficient and more brain like machine learning methods.

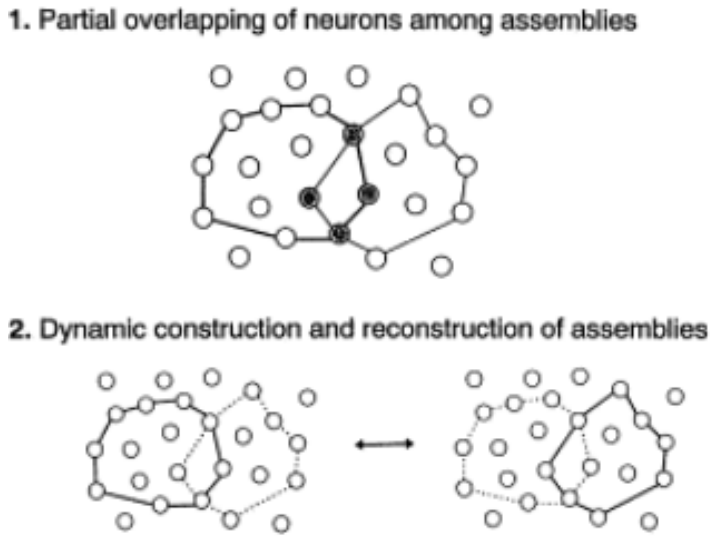



Figure 1.6: Visualization of some neuron assembly properties.

### 1.2.3 Neural Plasticity

Synaptic plasticity is a process that modifies the connectivity of neurons influenced by the amount of excitation between them . The neuroscientist Shatz described this as "cells that fire together, wire together" [30].More notably, if one of the cells is active systematically just slightly before the other, the first one's firing may have a causal relationship to the second one's firing, which can be later recalled by strengthening the wiring of connections : this is how neuroscientists define synaptic plasticity. Timing is important since it can reveal causality and also might serve as a temporal coding scheme on a millisecond time scale .

Long into the mid-1990s, Hebb's notion that coincident activity in linked neurons is what matters in plasticity dominated neuroscience research . It was realized that brain synaptic connections have mechanisms in place that should have made them especially sensitive to timing and it was confirmed in numerous studies [31] [32] [33], neuroscientists called this improved version of Hebbian Learning Spike Time Dependent Plasticity(STDP) .By potentiating those inputs that predicted their own spiking activity, a neuron contained in a neural network can select which neighboring neurons are worth listening to with STDP. The neuron in issue, on the other hand, pays less attention to

surrounding neurons that fail to do so. As a result, the sample neuron can integrate inputs with predictive strength and translate them into a meaningful predictive output, even if the meaning isn't completely understood by the neuron. As a result, STDP provides a very basic and elegant mechanism for properly connecting neurons in the brain [34] .

### 1.2.4 Oscillations

Oscillations occurring at different frequencies are considered as functionally relevant signals of the brain and research supports that event-related oscillations bridge the gap between single neurons and neural assemblies [35]. Research also hypothesizes that selectively distributed delta, theta,alpha, and gamma oscillatory systems act as resonant communication networks through large populations of neurons. They reflect the temporal synchronization of neuronal populations' behavior and have been implicated as a mechanism that selects subsets of neurons for further joint processing and eventual stimulus representation because they can display task or stimulus dependence [36] [37].

# Chapter 2

# Neuromorphic computing

Moore's Law, which predicted exponential growth in the number of transistors that could be made on a single microchip, has guided advances in microchip technology. The exponential time constant is short—it doubles every 18 months. Moore's Law has been implemented primarily through the reduction of transistor size, and as CMOS transistors become smaller, they become cheaper, faster, and more energy-efficient. Neuromorphic computing encompasses a wide range of information processing techniques, all of which are distinct from mainstream conventional computer systems by some degree of neurobiological inspiration. The theory underpinning neuromorphic computing can be traced back to Carver Mead's foundational work at Caltech in the late 1980s. This early work inspired others to continue developing neuromorphic devices, and the aforementioned breakthroughs in VLSI technology aided constant expansion in the size and functionality of neuromorphic devices [38].

Current general-purpose computers' digital designs provide the noise immunity and predictable behavior that the determinism Turing machine is known for.Biology foregoes determinism in favor of efficiency, which could be of interest to future computer engineers working on systems like robot vision systems, where absolute accuracy is impossible to achieve and energy efficiency is a top priority .Neuromorphic computing aims in some limited way to extract or mimic the complexity of the human brain and its principles of operation to more abstract methods that can be applied in a computing system of this type. Neuromorphics systems adhere to the principle of distributed computing, in other words having a large amount of small computing "cores" analogous to neurons connected into networks with some degree of

allowed connectivity flexibility.

## 2.1 Structural components and Architecture

Neuromorphic systems and implementations tend to deviate from the norm and instead of the use of typical transistors and circuit elements (i.e. resistors) that most electronic platforms use, exploit features of structural elements that present desirable behavior like memory. Although, implementations with typical electronic devices exist and presents satisfactory performance such as in [39].

One such electronic device used for neuromorphic development is the CMOS, Complementary metal–oxide semiconductor (fig. 3.1). A complete guide to CMOS is presented in [40]. CMOSs offer great advantages over typical circuit elements like low energy consumption and scalability and have been used in previous works like in [41], where the authors develop Temporal Neural Network with 32 million 7nm CMOSs covering in total 1.54 $mm^2$ with only 7.26 mW power with the capability of processing a 28x28 input image at 9.34 ns. Large-scale projects like IBM's TrueNorth chip, also utilize CMOS technology for neuromorphic computing.

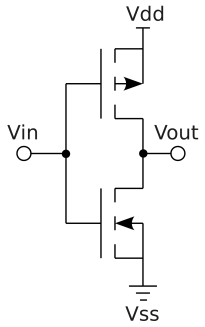
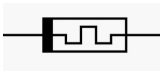
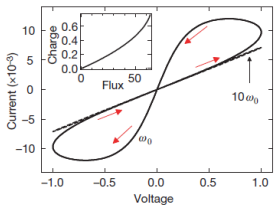



Figure 2.1: CMOS schematic diagram.

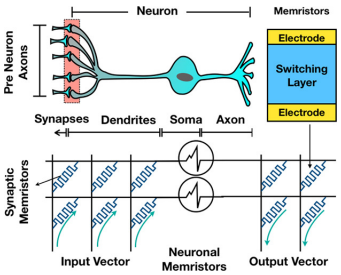
However, one of the most important devices for neuromorphic computing is the memristor [42]. The memristor is a two-terminal fundamental electrical component, that presents memory behavior by remembering its history. This property of non-volatility makes this element suitable for neuromorphic computing as it can mimic biological neuron behavior and it has repeatedly been used as a synapse, providing the feature of synaptic plasticity in the



(a) Memristor symbol




(b) Current-Voltage characteristics



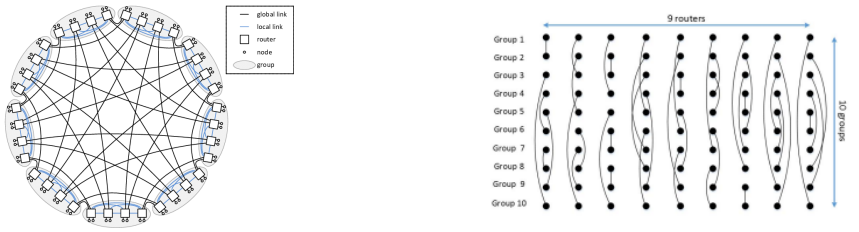

(c) Example of use of memristors in neuron simulation.

Figure 2.2: Memristor's features and uses

network [43], [44]. Due to the high interest of the research community in the use of memristors in neuromorphics, a wide variety of memristor-based implementations have been developed in order to increase its efficiency, such as the multi-memristive synapse proposed in the work of Boybat et. al. [45].

In continuation of the high utilization of space and energy of neuromorphic computers, the use of spintronics [46] was introduced in the field. Spintronics are devices that use both the electric and magnetic properties of electrons increasing the computational capabilities of each individual device with low energy consumption. In 2009 Wang et. al. in their work [47], proposed the development of spintronic memristors - spintronic devices with memristive effects. Torrejon et. al. in 2017 [48] showed that the biological oscillatory behavior of the neurons can be implemented with the use of nanoscale oscillators. As they show, such devices present noise resistance and can provide significant results in test runs. In the work of Grollier et. al. [49], the methods and results of spintronic memristors, oscillators and other spintronic-based methods are summarized and reviewed, providing a complete picture of spintronics in neuromorphics.

The property of synaptic plasticity is a major aspect of research in neuromorphics. A different way of inducing synaptic plasticity in the hardware is the use of suitable types of RAM (Random Access Memory) as synapses.




(a) Dragonfly Topology between 5 groups

(b) Custom Dragonfly Topology proposed in [55]

Figure 2.3: The difference of the Dragonfly Topology (a) and the custom one proposed by the authors of [55] (b). In (b) the nodes of each group are connected through a bus that is not drawn for simplicity.

The most noteworthy types of RAM used in neuromorphics are: Resistive RAM (RRAM), Conductive-bridging RAM (CBRAM), Phase-change RAM (PRAM or RCM) and Magnetic RAM (MRAM) which are reviewed and studied in detail in the following papers [50], [51], [52]. In the following work [53], Suri et. al. showed that RCM can mimic behaviors of biological synapses such as Long Term Potentiation (LTP) and Long Term Depression (LTD) plasticity effects, which are important features for neuromorphic computing.

The development of ideal structural elements for neuromorphic computing is very useful, so is progress in the architectural design of such networks. Important features are studied constantly such as expandable networks [54], which offer scalability with no added latency or topologies that increase the performance of the network. In [55], the authors propose a new Network-on-Chip topology, based on the dragonfly topology, which offers faster information processing to the network.

Spatial and energy efficiency is a major aspect of neuromorphic computing, which makes the technology attractive in the research community, and it is constantly developing. With increasing research in microelectronics and nanotechnology, a lot of devices enter the game to overcome their competitors. Nanofibers, electrochemical transistors, nanotubes, quantum dots and much more are under the microscope to improve performance and make neuromorphic computing state-of-the-art.

Neuromorphics propose a new way of computing. Using the appropriate units that show desirable behavior, such as non-volatile memory, the processing power is rapidly increasing, while the targeted purpose of such systems, mainly in Spiking Neural Networks, allow the use of architectures other than the von Neumann - the most common one used in modern computers, to

specialize and exploit features that the aforementioned architecture cannot provide.

## 2.2 Neuromorphic computers

In recent years, a number of large-scale neuromorphic systems have emerged, utilizing the massive transistor resource now available on a single microchip and, in one case, an entire silicon wafer. The technology's capabilities combine with scalable architectures to allow neuromorphic capabilities to scale up to support neural networks with millions of neurons and billions of synapses. Modelers can now consider developing models of the entire brains of creatures ranging from insects to tiny mammals, or major sub-areas of the human brain. The same systems also provide platforms capable of supporting new scales of cognitive architecture. Some of the most notable examples are the following.

### IBM TrueNorth

The IBM TrueNorth chip is based upon distributed digital neural models aimed at real-time cognitive applications.The chip is a very large, 5.4 million transistor 28 nm CMOS chip that incorporates 4096 neurosynaptic cores where each core comprises 256 neurons each with 256 synaptic inputs .The cross-bar outputs couple into the digital neuron model, which implements a form of integrate-and-fire algorithm with 23 configurable parameters that can be adjusted to yield a variety of different behaviors, and digital pseudo-random sources are used to generate stochastic behavior by modulating synaptic connections, neuron threshold, and neuron leakage [56].The outputs of each core's neuron spike events follow individually-configurable point-to-point paths to the input of another core, which can be on the same or a different TrueNorth chip.When a neuron's output must link to two or more neurosynaptic cores, the neuron is duplicated within the same core (see Fig 2.4). The digital model's deterministic nature ensures that all replicants produce identical spike trains [38].

### SpiNNaker

The SpiNNaker project [57] has developed a massively parallel digital computer whose communication infrastructure is inspired by the goal of modeling

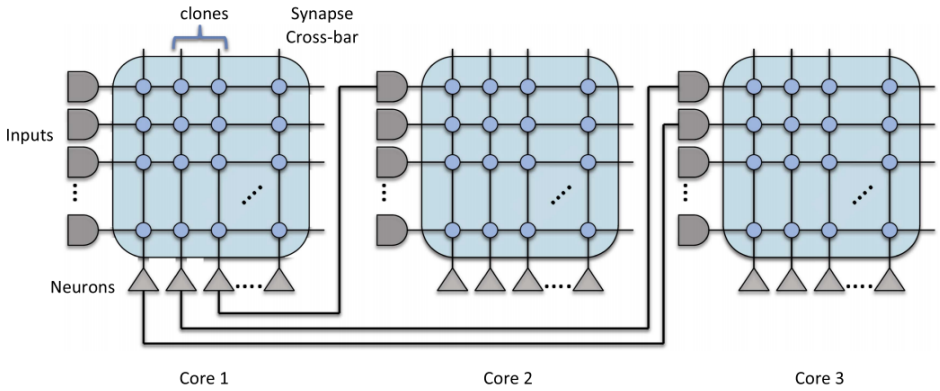



Figure 2.4: TrueNorth communications are built on point-to-point links that carry spikes from a single neuron to a single neurosynaptic core, where the spikes can connect to any or all of the core's 256 neurons. For example here,Core 1's leftmost neuron links to core 3 . Core 1's 2nd and 3rd neurons duplicate each other to link to cores 2 and 3, and each makes one connection.

large-scale spiking neural networks in biological real time with connections similar to a brain's. The largest SpiNNaker system currently in use has 1 000 000 cores. However, spiNNaker is unlike other neuromorphic systems . It uses small integer cores (custom chips) intended for mobile embedded applications .

SpiNNaker's communications fabric is designed for sending large amount of small data packets(i.e. neuron spikes) to many destinations according to statically configured multicast paths [58]. The design of SpiNNaker is based around a small plastic 300 bga (ball grid array) package which incorporates a custom processing chip and a standard 128 Mbyte SDRAM memory chip. The processing chip, designed on a 130 nm CMOS technology, contains 18 ARM968 processor cores, each with 32 Kbytes of instruction memory and 64 Kbytes of data memory, a multicast packet router, and sundry support components [59] [38].

### Neurogrid

### BrainScaleS

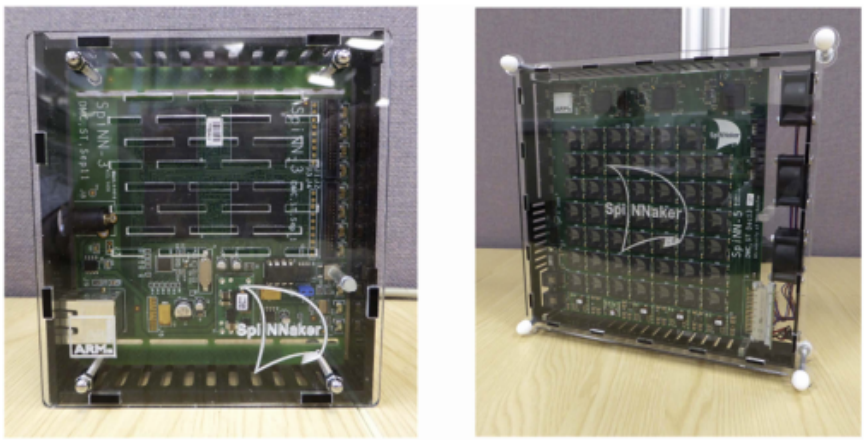



Figure 2.5: SpiNNaker systems on a small scale. The 4-node (72-core) system (left) is powered by a USB connector and is ideal for learning and small tasks, such as robots. The 48-node (864-core) system (right) is the basic building block of the larger computers and can be employed for larger applications.

| Platform: | Human brain | Neurogrid | BrainScaleS | TrueNorth | SpiNNaker |
|---|---|---|---|---|---|
| Technology: | Biology | Analogue, sub-threshold | Analogue, over threshold | Digital, fixed | Digital, programmable |
| Microchip: | | Neurocore | HiCANN | | 18 ARM cores |
| Feature size: | 10 $\mu$m[a] | 180 nm | 180 nm | 28 nm | 130 nm |
| # transistors: | | 23 M | 15 M | 5.4 B | 100 M |
| die size: | | 1.7 $cm^2$ | 0.5 $cm^2$ | 4.3 $cm^2$ | 1 $cm^2$ |
| # neurons: | | 65 k | 512 | 1 M | 16 k |
| # synapses: | | ~100 M | 100 k | 256 M | 16 M |
| power: | | 150 mW | 1.3 W | 72 mW | 1 W |
| Board/unit: | | PCB | 20 cm wafer | PCB | PCB |
| # chips: | | 16 | 352 | 16 | 48 |
| # neurons: | | 1 M | 200 k | 16 M | 768 k |
| # synapses: | | 4 B | 40 M | 4B | 768 M |
| power: | | 3 W | 500 W | 1 W | 80 W |
| Reference system: | 1.4 kg | | 20 wafers in 7 × 19″ racks | | 600 PCBs in 6 × 19″ racks |
| # neurons: | 100 B | | 4 M | | 460 M |
| # synapses: | $10^{15}$ | | 1 B | | 460 B |
| power: | 20 W | | 10 kW | | 50 kW |
| Energy/connection: | 10 fJ | 100 pJ | 100 pJ | 25 pJ | 10 nJ |
| Speed versus biology: | 1× | 1× | 10 000× | 1× | 1× |
| Interconnect: | 3D direct signalling | Tree-multicast | Hierarchical | 2D mesh-unicast | 2D mesh-multicast |
| Neuron model: | Diverse, fixed | Adaptive quadratic IF | Adaptive exponential IF | LIF | Programmable[b] |
| Synapse model: | Diverse | Shared dendrite | 4-bit digital | Binary, 4 modulators | Programmable[c] |
| Run-time plasticity: | Yes! | No | STDP | No | Programmable[d] |

Figure 2.6: Overview of neuromorphic chips and a comparison with the human brain.We still have a long way to go to reach the human brain.

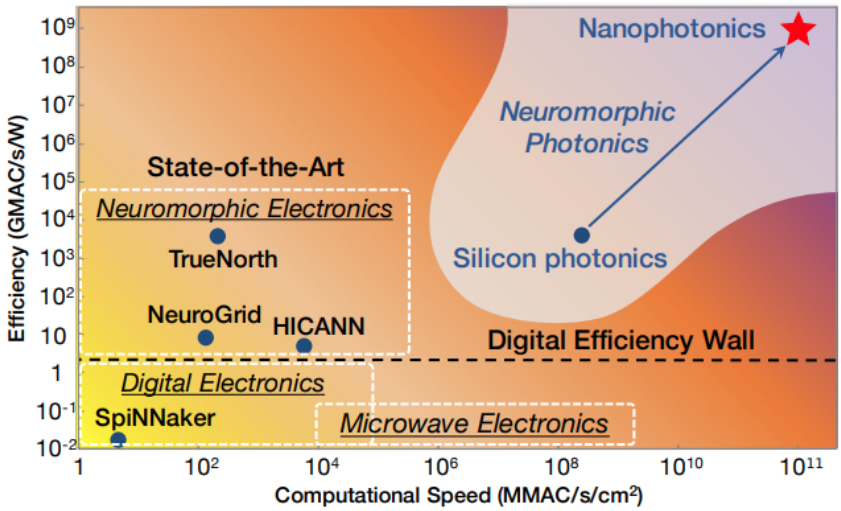



Figure 2.7: Comparison of neuromorphic hardware platforms . The superiority of neuromorphic computing systems in terms of efficiency is apparent.Future photonic neuromorphic systems might offer substantially better performance but photonics fall out of the subject of this thesis. [60]

# Chapter 3

# Neuron Models

In this chapter, we will shift our attention from the biological brain to the artificial one. Spiking Neural Networks differ from the common Artificial Neural Networks that have been developed throughout the last century, as stated before, mainly in the temporal dimension of the input data. This difference creates the need for the development of neuron models such that they can decode and extract information from the temporal data. Moreover, most of the datasets for training and testing ANNs that currently exist, consist of non-temporal data. In the literature, many such models have been developed for the implementation of SNNs, some of which will be reported in this chapter.

## 3.1 The Hodgkin - Huxley model

In 1952, Hodgkin and Huxley published 4 papers regarding how the neurons work [61]. They performed experiments on a giant axon of a squid and developed the following model. Through their tests, they found that the ionic movement from tree ions, Sodium ($Na^+$), Potassium ($K^+$) and a leak current mainly consisting of Chlorine ions ($Cl^-$), controlled by two voltage-dependent channels (Sodium and Potassium channels) are responsible for the current flow of the neuron. While the neuron is at rest, inside the neuron, a high concentration of negatively charged ions creates a voltage difference with the exterior of the neuron, which is positively charged. When the voltage reaches a certain threshold, Sodium and Potassium pumps open, moving ions in and out of the cell respectively. This ionic movement results in a current

that moves as a spike through the axon to the next neuron.

### 3.1.1 Mathematical Representation

The Hodgkin-Huxley model is represented as an electrical circuit shown in figure 3.1. The voltage denoted as $V_m$ represents the voltage across the cell membrane. We can see that an input current $I_m$ can charge the capacitor $C_m$ or it can leak through the rest of the channels. The potential of each of the ions is different so it is used one different battery for each one of the ions. It is worth noting that the Sodium battery $E_{Na^+}$ is oriented in reverse compared to the rest of the ion batteries and this is justified as the Sodium ions move out of the membrane. The arrows in the resistors of the Sodium and Potassium channel denote that these resistors are though as non static values and describe the pumps of the biological model.

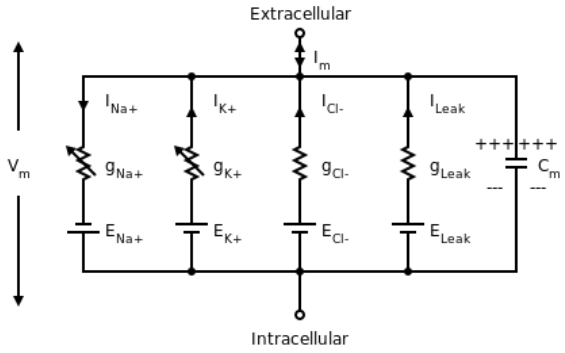



Figure 3.1: The Hodgkin-Huxley model represented as an electrical circuit.

In order to analyze the circuit, the static values of the Chlorine channel and the leak channel can be calculated with Kirchhoff's law as one channel, named Leak channel, with a steady valued battery and resistance. Applying the Kirchhoff's current law in the circuit we get the following equation:

$$I(t) = \sum_k I_k$$

Replacing each current equation with its voltage equivalent, arises a first order differential equation due to the capacitor $C_m$. For each of the Sodium,

Potassium and Leak channels, the corresponding conductance has to be calculated. Hodgkin and Huxley found that two types of pumps are responsible for the movement of Sodium ions and one pump type for the Potassium. For this reason, they added the parameters m, h and n to describe the control of the pumps in the ion movement. Through numerical experiments they arrived at the following result describing the potential of the cell membrane

$$C_m \frac{dV_m}{dt} = I_m - (g_{Na^+} m^3 h(V_m + E_{Na^+}) + g_{K^+} n^4 (V_m - E_{K^+}) + g_L(V_m - E_L)) \tag{3.1}$$

where the values m, h and n are voltage dependent and are described by the following differential equations:

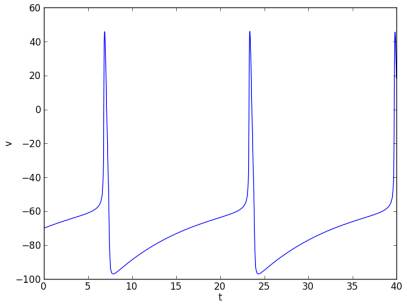
$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) + \beta_m(V_m)m \tag{3.2}$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) + \beta_h(V_m)h \tag{3.3}$$

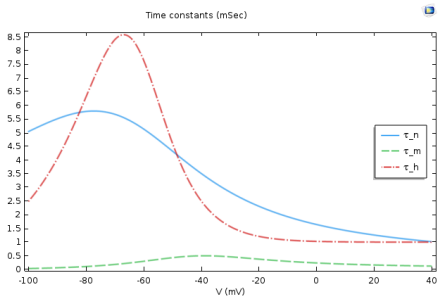
$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) + \beta_n(V_m)n \tag{3.4}$$

The $\alpha_m(V_m)$, $\alpha_h(V_m)$, $\alpha_n(V_m)$, $\beta_m(V_m)$, $\beta_h(V_m)$, $\beta_n(V_m)$ are voltage dependent functions that define the behavior of the m, h and n variable and, in total, the cell membrane potential. Solving each equation, we get an exponential solution with an exponent constant $\tau_m$, $\tau_h$ and $\tau_n$ respectively. In figures 3.2a and 3.2b is shown the membrane voltage that occurs through the above set of equations and the values of the time constants as a function of potential. This difference indicates the existence of fast and slow ion gates.

Given exact values at the voltage-dependent variables of them, n and h functions, we can simulate the voltage of the cell membrane and study its behavior. In their work [62] Nelson M. and Rinzel J. used a GENESIS tutorial squid [63] to generate multiple forms of a spike. The Hodgkin Huxley model has been developed since 1952 to adapt to the findings of the behavior of neurons and this book [2] presents extended information about the biological model of the neuron and the mathematical adaptation of the Hodgkin Huxley model.




(a) Hodgkin-Huxley model membrane's voltage trajectory




(b) time constants as a function of potential

Figure 3.2: Hodgkin-Huxley model dependencies

### 3.1.2 Implementations and uses

The Hodgkin Huxley model pioneered the field of neuronal dynamics in the aspect of neuroscience but it also initiated the research and development of Spiking Neural Networks. From an engineering perspective, there have been many successful attempts to implement a neuron using the aforementioned model in hardware with FPGA as in [64] which results in a possible communication between electrical signals from a living organism to an artificial structure and, with a development of a SNN, cooperation between them. There have also been developed multilayered Spiking Neural Networks with the Hodgkin Huxley model of the neurons and been trained for tasks like edge detection and pattern classification as shown in this [65] and this [66] research work respectively with promising results.

### 3.1.3 Model weaknesses

Despite its innovative description of a neuron, the model has been criticized for its efficiency in describing the complex behavior of the many different types of neurons. These criticisms arise from weaknesses of the model such as its inability to account for events that can affect the neuron's state [67]. From engineering perspective, its main drawback for the development of complex SNNs is its high computational complexity. It has been proven [68] that the Hodgkin Huxley model can be described, with sufficient accuracy, as a single-variable threshold model. Engineers shifted their attention to developing

simpler threshold models so as to be able to create more complex structures which led to the development of neuron models such as the ones that follow.

## 3.2 Leaky Integrate-and-Fire Model: LIF

The LIF model was first proposed in 1907 [69] and presents the neuron as a leaky integrating unit. The LIF model, in contrast with the HH, is a threshold model, that is, it emits an output spike when the input voltage reaches a predefined threshold. The equivalent electrical circuit of the model is a RC circuit with an input current $I_{inject}$. The input currents are typically spikes ($\delta$-Dirac like functions) that, due to the capacitor C, increase the voltage of the unit by the same value. The resistor, or as shown in Figure 3.3 the conductor $g_{leak}$, is responsible for the neuron's voltage leakage, while the source $E_m$ accounts for the neuron's voltage in idle state. When the threshold is reach, the unit emits a spike and the voltage returns to the initial(idle) state.

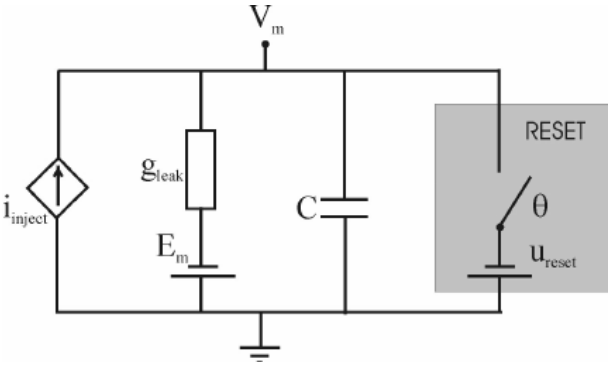



Figure 3.3: The Leaky Integrate-and-Fire model as an electrical circuit

### 3.2.1 Model Analysis

Analyzing the circuit the following equation is derived, describing the potential of the unit through time. The solution of the homogeneous equation ($i_{inject}$=0) is an exponential decay. The constant $\tau_m = C/g_{leak}$ is the time constant of the exponential decay and is determined by the aforementioned

factors, the conductor and the capacitor of the circuit. Figure 3.4 shows the response of the model in a spike train input.

$$\tau_{\mathrm{m}} \frac{dV_{\mathrm{m}}(t)}{dt} = -(V_{\mathrm{m}} - E_{\mathrm{m}}) + \frac{i_{\mathrm{inject}}}{g_{\mathrm{leak}}} \tag{3.5}$$

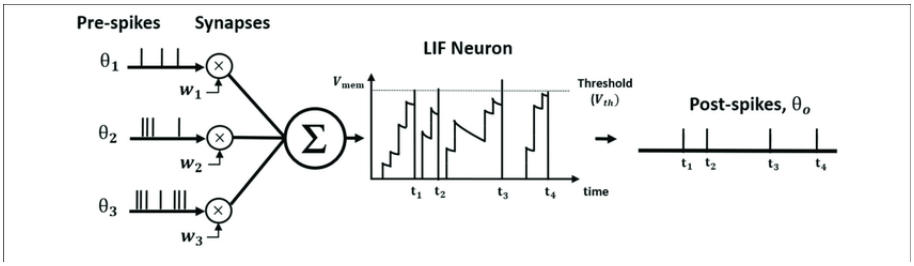



Figure 3.4: Voltage of neuron's potential and its response. Each of the input signal is multiplied by its weight and then summed to act on the neuron's potential. When the neuron's membrane potential reaches the threshold, it produces a spike and the voltage is reset to its idle state.

### 3.2.2 Uses and Limitations

The simplicity of the model has drawn the attention of researchers both for software and hardware implementation of the neuron. It has been used for a wide variety of usages. In 2003 the leaky integrate-and-fire model was used to model the cochlea and detect sound features [70], while more common, for ANNs, tasks, such as pattern recognition and image segmentation, have been tackled using either single LIF model neurons (for pattern recognition) [71] or SNNs that consists of such neuron models (image segmentation) [72]. Doutsi E. et al. in their work [73] used the LIF model to transform the input signal to spike trains. This extended research over this model led researchers to view the model itself as a subject for study and much work has been done in improving the model itself, like in [74]. From the point of view of the hardware implementation, SNNs based on the model mentioned above have already been proposed or developed and trained for various purposes such as [75] and [76], exploiting the advantages of SNNs. The increasing

research development of electronic components, like memristors, that favor the development of such networks [77], has created a explode in the creation of neuromorphic, and even photonic architectures [78], that promise low power consumption [75], [79], and high compacted architectures [80] for developing LIF model-based SNNs.

The drawback of the model emanates from its simplicity. It has been proposed that, compared to the HH model, the LIF model could be less tolerant to noise while resembles less the biological neuron [81]. Furthermore, compared to the results of the state-of-the-art ANNs and machine learning algorithms, SNNs do not always present satisfactory results, such as in this study [82] where a Leaky Integrate-and-Fire SNN model is compared with an SVM.

## 3.3 Izhikevich Model

### 3.3.1 Model definition

In 2003 Eugene M. Izhikevich published an article proposing a new model of neurons [83]. The goal was to provide a model with mathematical simplicity and biological plausibility, combining features from the Leaky Integrate-and-Fire model and the Hodgkin-Huxley model. The resulting model would be useful to model biological neurons while could be easily implemented, from an engineering perspective. The mathematical description is presented in the following equations. Like the LIF model, it is a threshold model, where different values of the parameters could be inputted to simulate a variety of neurons.

$$\frac{dv}{dt}(t) = 0.04v^2 + 5v + 140 - u + I(t) \tag{3.6}$$

$$\frac{du}{dt}(t) = a(bv - u) \tag{3.7}$$

$$if\ u \geq 30mV\ then \begin{cases} v = c \\ u = u + d \end{cases} \tag{3.8}$$

The $v$ value represents the membrane potential of the neuron while the $u$ value represents a membrane recovery variable. In his work, Izhikevich showed that manipulating the a, b, c and d parameters, can produce a wide

variety of neuronal responses. Later, in 2004 he stated [84] that the Izhikevich model ranks high in combining high biological plausibility and low implementation cost, amongst the existing models of its era.

### 3.3.2 Implementations

Indeed, in the last decade, the Izhikevich model has been implemented in integrated circuits exploiting recent developments of the field. Multiple techniques that compete with each other such as MNIN [85] and CORDIC [86] promise lower computational cost and error performance. The model has been tested in classical Artificial Intelligence tasks, either implemented in its own hardware, such as in this paper [87], where an FPGA was developed to represent an Izhikevich model based Spiking Neural Network and was tested in character recognition, or in software that simulates the response of the model. Single neurons have been tested in non-linear pattern recognition problems [88] showing the ability of an individual neuron in classifying multiple patterns. One clever exploitation of this ability is presented here [89] where the multilayer perceptron network used for the classification of a typical CNN is replaced by an Izhikevich neuron. As a result, this network achieves similar scores with typical CNNs while reducing the training time.

### 3.3.3 Criticism

Despite the promising statements from the creator of the model, in recent years, it has been criticized. Reviews and comparisons with other models (such as LIF and HH) show minuscule, if any, advantages. In 2014, Michael J. Skocik and Lyle N. Long compared the computational cost of the Izhikevich, LIF and Hodgkin-Huxley model using multiple arithmetic methods (so as to find the best implementation of each model) [90]. Their results showed that the Izhikevich model is comparable with the HH model in terms of computational cost, while the LIF model is, as expected the best out of the three. In 2017, Sergio Valadez-Godínez et al. [91] compared the same neuron models in terms of accuracy and cost if different fire rates. They also concluded, that in most cases, the Izhikevich model gave bad results, being less efficient than the HH model, while being more computationally expensive than the LIF model. A more recent study of 2020 from the same researchers [92] presents and sums up the problems of the Izhikevich model that been proposed in the bibliography.

## 3.4 Spike Response Model - SRM

### 3.4.1 Model definition

The Spike Response Model was developed, as an idea, in a series of papers in the last decade of the 20th century, but the name was first introduced in this paper [93] of 1993. In this paper, Wulfram Gerstner et.al. present the mathematical model of the SRM neuron along with simulation results. The model resembles the Integrate-and-Fire model (IF model), with the difference that the membrane potential is dependent on various linear kernels that act on the incoming spikes. Furthermore, the threshold in this model is time-dependent in contrast to the Integrate-and-Fire model. In this book [2], the authors dive into an extensive description and explanation of the SRM and its similarity with the IF model. The following set of equations (3.9), (3.10) describe the membrane potential and the threshold value of the model, respectively. The aforementioned kernels $\eta(), \kappa()$ and $\theta_1()$ can be interpreted as linear response filters acting on the incoming or the postsynaptic spikes. In more detail, the $\kappa$ kernel is the linear response of the membrane for the incoming spike and it integrates the input current through time. The $\eta$ kernel corresponds to the action potential of the neuron. This is better understood by the fact that the kernel affects the membrane potential and it is dependent on the $t^f$ variable, which resembles the firing time of the neuron. The kernel has to contain even the negative overshoot of the neuron's potential. The $\theta_1$ kernel describes the neuron's short-term plasticity and changes the threshold value of the neuron (in most cases) after the neuron fires.

$$u(t) = \sum_{t^f} \eta(t - t^f) + \int_0^\infty \kappa(s) I(t - s) ds + u_r est \tag{3.9}$$

$$\theta(t) = \theta_0 + \sum_{t^f} \theta_1(t - t^f) \tag{3.10}$$

The model can also be interpreted as closed loop system where each kernel is a finite impulse response filter (FIR filter), the input of the system is the input current of the neuron, the output is the spike train that the neuron produces due to its input and the system's state itself is the membrane potential. The following image, taken from the book [2] gives an optical representation of the system mentioned above.

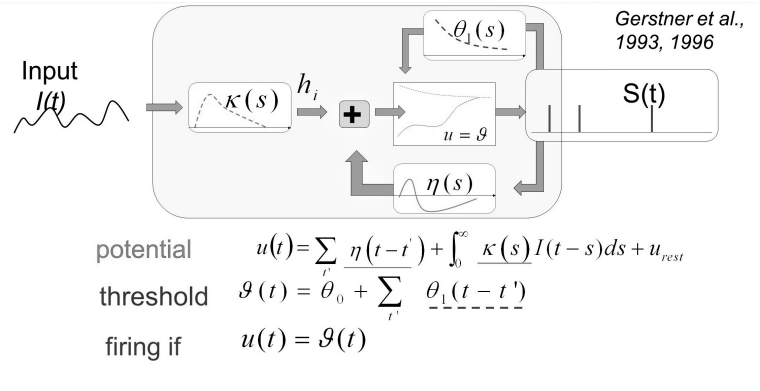



Figure 3.5: SRM diagram. Each filter corresponds to a kernel that describes the mathematical model. The system represents the membrane potential and the output, the spike train of the neuron

### 3.4.2 Model capabilities

The abstract nature of the model due to the non-strictly defined kernel functions, gives the ability to implement a variety of neuron types and action potential responses and fine-tune the variables to create models that mimic the functionality of real neurons. In 2003 Reanut Jolivet et.al. showed that the Spike Response Model could predict with high accuracy the response of real neurons, given the same input [94]. Improvements of the model have been proposed such as in [95] which make the model more biologically plausible

The Spike Response Model is said to be a generalization of the Integrate and Fire model. That said, it may have been expected that neural networks based on this neuron model would draw the attention of Artificial Intelligence engineers. Although some works have used such networks to tackle tasks like Breast Cancer Diagnostic, as shown in [96], with impressive results, the SRM does not attract the interest of the engineering community. As a result only a few hardware implementations exist, such as [39], while the model itself stands as a tool mainly from biological perspective.

## 3.5 Stochastic and Probabilistic Neuron model

### 3.5.1 Stochastic Resonance

All neuron models that have been presented were deterministic systems, described by fixed valued equations and deterministic inputs. As a result, by definition, knowing the initial state and the input signal, the response of the model can be deduced. This fact can be useful as it helps understanding important features as the stability of the model. However, it has been suggested that adding stochasticity in such biological systems can be useful since it gives the ability for weak signals to be detected. This idea is called stochastic resonance and was presented in 1981. Stochastic resonance is said to be present when the following three conditions exist in the system. First, there has to be an activation barrier, such as the threshold value presented in the previous models. Moreover, weak and periodic signals, such as the input spike-train must be applied. Finally, an input noise has to be applied in the input signal. Stochastic resonance is extensively analyzed in this work [97]. This phenomenon is not only a useful tool from an engineering perspective, but is also observed in biological systems [98]. Consequently, there has been an extensive research in intentionally adding stochasticity (such as by applying a noisy input) or probability in neuron models, so as to exploit the effects of stochastic resonance.

### 3.5.2 Stochastic noise and diffusion

Stochasticity, as mentioned before, does show up in biological models. Stochasticity of the system can be caused by a noisy input, as presented above, with the existance of stochastic resonance in deterministic models, such as in [39]. Nonetheless, the model itself can be guided by stochastic processes and probability embedded in its parameters. It is known that the opening and closing of ion pumps in the neurons is guided by some amount of stochasticity embedded in the system. To tackle this behavior, stochastic models of the Hodgkin-Huxley model have been proposed [99] that treat the opening and closing of the ion pumps as non-deterministic events. While such models can be useful from biological perspective, they do not serve a remarkable purpose in developing trainable large-scale spiking neural networks due to their complexity. For such usage, non-deterministic model of other, simple models have been developed such as the LIF model and the Spike Response Model.

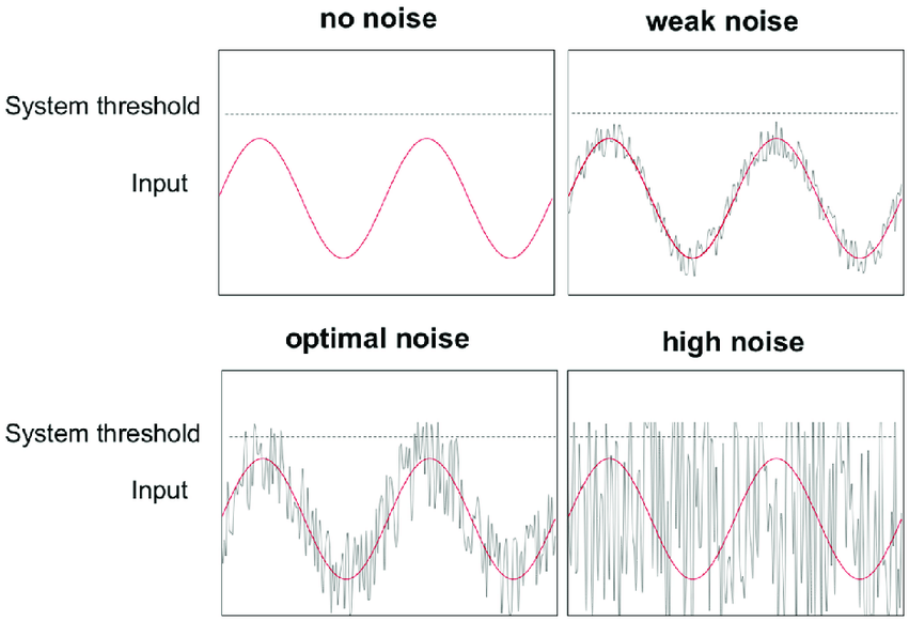



Figure 3.6: Stochastic Resonance. An input noise is added to the weak signal. As a result, the new noisy input can exceed the threshold and the activation occurs. The intensity of the added noise plays an important role in the output and has to be handled carefully. A low intensity noise will not cause the input to exceed the threshold, while a high intensity one will dominate the input signal and the information will be lost.

The non-deterministic behaviour is usually added in parameters that correspond to the three phases of the neuron: the presynaptic, the membrane and the postsynaptic. In his work [100], N.Kasabov defines a probabilistic neuron model where each of the three aforementioned stages contribute with a probability for the spike arrival, the spike contribution to the neuron membrane and the spike generation. Developed network with stochastic neurons for pattern recognition tasks show features that deterministic SNNs lack of. A common conclusion that is typically mention is the robustness of the network compared with the deterministic one, as stated in this paper [101] where the probabilistic LIF neuron model was used, and in this one [102] where this time a stochastic SRM with a spike generation probability was used. In this paper [103] multiple non-deterministic models are compared to the corresponding deterministic, showing robustness and explicit behaviour.

Another approach in adding stochasticity to the model is the infusion of stochastic diffusion. The most common model that uses the diffusion process as a stochastic neuron model is the Ornstein-Uhlenbeck model [104]. Another model that uses the same idea is the Feller model. As a result of their unique and complex nature, these types of models have attracted the interest and parameter estimation has been extensively studied over them, as presented in [105] and in [106]. However, it should be mentioned that only the surface has been scratched of the stochastic neuron model family. There have been developed a wide variety of neuron models that exploit the stochasticity in different ways and have it act on different aspects of the model. Two more such models are presented here, the Galves–Löcherbach model [107] and the stochastic Fitzhugh-Nagumo model [108], left for the reader to dive into.

### 3.5.3 Model applications and potentials

As presented above, stochasticity in neuronal networks could be a useful tool to explore unnoticed behaviors of the systems, but one could argue that coding and implementing such systems in hardware is inefficient and probably computationally expensive. However, stochastic resonance has to do with adding noise to the input. As a result, by carefully manipulating the electronic components of the hardware, the naturally created noise can be added to the input and give the desired noisy input spike-train. Such devices, like stochastic memristors, have been tested in their ability to control the noise of the device and create fully stochastic switching devices [109], and such devices have been used in developing hardware components with embedded

stochasticity for spiking neural networks [110]. Moreover, memristors are not an one-way street in developing stochastic SNNs. Different components, such as the avalanche diode [39], can be manipulated to produce the desired noisy input, or even to exploit smart architecture in FPGAs and develop stochastic SNNs with fully deterministic components, where stochasticity is digitally added [111].

One topic that stochastic SNNs (and ANNs in general) show potential is finding fast approximate solutions to NP-complete problems. Biological systems show a high ability in finding good enough solutions in constraint satisfaction problems. Knowing this, stochastic SNNs have been developed to mimic such behavior with satisfying results [112]. Another promising use of stochastic spiking neural networks is their use as a reservoir computer or, in the case of spiking neural networks, liquid state machines. Liquid state machines(LSM) consist of a large collection of neurons that are randomly connected to each other and each one receives input spikes through time from an external source and other neurons in the LSM. Due to their architecture LSM have high capabilities in computing a large variety of non-linear functions and the added stochasticity can even further improve the framework. Such stochastic LSMs (sLSM) have already been tested against their deterministic LSM [113] and it has been deduced that they can perform better than the deterministic ones. This framework has already been used successfully for developing a SNN for EEG classification [114].

# Chapter 4

# Spike Information Processing

Most ANN training and testing datasets available today consist of non-temporal data. Such datasets cannot be given as input to an asynchronous model such as an SNN. To overcome this problem, several encoding algorithms have been developed, some of which will be presented in this chapter.

## 4.1 Information Representation

The added temporal dimension that is embedded in SNNs, gives rise to the need of encoding the input data in such ways that the Network can process. There are two main encoding schemes to represent the given information into spiketrain, rate coding and pulse or temporal coding. The choice of data encoding constitutes an important decision in the design of the Neural Model, as it gives the ability to develop models with different learning rules, compatible with the encoding scheme that was chosen, and form models in a wide spectrum of biological plausibility and engineering computability.

### 4.1.1 Rate Code

Rate coding is the encoding scheme where the information is encoded to the number of spikes emitted within a time window. The firing rate of a neuron is calculated as the number of spikes the neuron produces within a given time window. There are many encoding algorithms that are based on the rate coding scheme that count the firing rate of single neurons or populations of neurons during one or multiple time windows. However, calculating the aver-

age firing rate of a population of neurons with similar properties over a time window may not be very useful in representing or extracting information. This method of representing information is widely used due to its simplicity and low complexity. However, focusing on the number of spikes in a time window and not the exact time that each spike occurs, information that is encoded in the exact timing of the spike has little contribution or is even lost. Furthermore, while there are some cases that the firing rate of neurons actually represents information in biological systems, as shown here [115], it is commonly accepted that rate coding shows little biological plausibility. In this work [116], Richmond and Optican showed that initial layers of neurons may be highly correlated with firing rate, but for "deeper" layers, there was little correlation between spike count and information representation.

### 4.1.2 Temporal Code

Temporal coding is based on the idea that information is encoded in the exact time of the spike emitted or the time difference between spikes. Temporal coding is shown to carry more information like the timing of the spike carries information on its own, increasing the efficiency of neural connections [117]. The information density contained in temporal coding schemes has been repeatedly shown superior over rate coding schemes, with a remarkable result being stated in this paper [118], where the use of Temporal coding in mapping a typical ANN to an SNN uses up to 10 times fewer operations compared to rate coding. Temporal coding provides another very important feature that its possible absence makes the study of learning algorithms in Spiking Neural Networks difficult, differentiability. In this research [119], the authors show that the input-output relation of s SNN with temporal encoding scheme is differentiable almost everywhere. Using this property, variations of back-propagation are available as learning rules, such as SpikeProp [120] (see also Chapter 4.3.1). Moreover, as this work proves [121], temporal coding enables Hebbian learning through Spike-Time Dependent Plasticity (see Chapter 4.3.2), while rate coding requires the use of forgetting function.

Despite the differences of the aforementioned coding schemes, one has not yet prevailed over the other, as engineers exploit the advantages of each one focusing either on computational efficiency or biological plausibility. This has led the research field to focus on developing universal SNNs that are able to extract information and be trained with either Rate or Temporal

encoded input, by developing algorithms that are compatible with both types of encoding formats by manipulating either scheme type, as previously shown in [121] where a forgetting function was developed to handle rate coded information and developing a universal Spiking Neural Network with the ability to learn with both types of encoded input, applying small changes to learning algorithms and adapt them to each learning scheme [122] or even mix the two schemes into new promising approaches, as shown in this work [123], where the author proposes such a novel encoding scheme with promising results, despite being in early stages of its development.

## 4.2 Encoding and Decoding Spikes

Over the past few years, a large amount of data has been collected and is used in the field of Artificial Intelligence. Such data consists mostly of continuous or discrete non-temporal values. As a result, such datasets cannot be used as input for Spiking Neural Networks without being encoded. Such problems may occur even with temporal data such as sound, where an analog signal is continuous through time. The following encoding algorithms have been developed to tackle these problems and convert such signals into spiketrains to be used as input data for Spiking Neural Networks.

### 4.2.1 Poisson Spike Generation

Poisson Spike Generation is a widely used Rate Coding scheme to convert analog values to spiketrains in a given time window. The value/intensity of the signal denotes the "frequency" or the normalized number of spikes in the given time window. Nonetheless, instead of fixing a frequency of the spikes in the spiketrain, generating a spike every $\tau$ seconds, it uses a homogeneous poisson process [124]. In this scheme, the normalized analog value represents the probability $r$. The main idea of Poisson Spike Generation is that the probability that a spike is generated during a time interval $dt$ is around $rdt$. We can also calculate the probability that the next spike occurs before time $\tau$ is $1 - e^{-r\tau}$

$$P(1\ spike\ during\ dt) \approx rdt \tag{4.1}$$

$$P(next\ spike\ occurs\ before\ \tau) = 1 - e^{-r\tau} \tag{4.2}$$

The average number of spikes emitted in the interval $(t1, t2)$ is calculated in the equation (4.13) and the probability of having n spikes during the interval $(t1, t2)$ is calculated in the equation (4.14). The result calculated is a poisson distribution, which gives its name to the encoding algorithm. This result guarantees that, while the exact timing of the spikes will not be known or even be the same for the same input values, the number of spikes will be almost proportional to the normalized analog input probability. That means that a higher input value will generate more spikes rather than a lower one in a given time window and the ratio of these values will be near the ratio of their spike count in the spiketrain.

$$\langle n \rangle = \int_{t1}^{t2} r dt = r\Delta t \tag{4.3}$$

$$P(n\ spikes\ during\ \Delta t) = e^{-r\Delta t}\frac{(r\Delta t)^n}{n!} \tag{4.4}$$

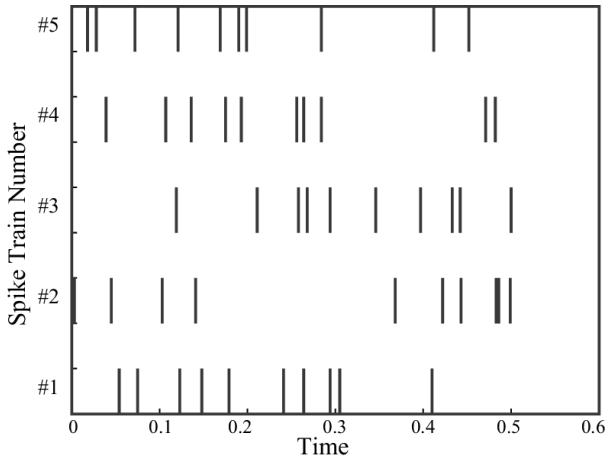



Figure 4.1: Multiple spiketrains generated with the same probsbility $r$ over the same time-window

This encoding algorithm is proven to be useful as it encodes the information in the fire rate over the time window, but also adds a noisy effect, the randomness of the exact timing of the spike (fig. 4.7). In this way, it adds robustness to the network since it learns to extract the information despite the probabilistic spiking and also shows increased biological plausibility, as it has been found that noisy rate encoded input is used in biological systems

and the brain. In this work [116], Richmond and Optican showed that initial layers of neurons may be highly correlated with firing rate, but for "deeper" layers, there was little correlation between spike count and information representation.

### 4.2.2 Rank Order Coding (ROC)

Rank Order Coding is a Temporal Encoding algorithm in which, the signal information is contained in the order of the firing neurons [125]. Given a set of five neurons A, B, C, D and E, the order they fire carries the information where the first spike emitted is considered as most important (see fig 4.8). An encoded stimulus could look like this B¿A¿D¿E¿C which denotes that neuron B fired first, neuron A second etc. while B carries the most information followed by A and so on. It is an easily implemented algorithm that gives the ability for N neurons to encode N! different values (there are N factorial different arrangements of N objects) and as such encode $log_2(N!)$ bits of information in the given time window, assuming each neuron can fire only once, a dramatic increase in information capacity compared to $log_2(N)$ of rate coding algorithms.

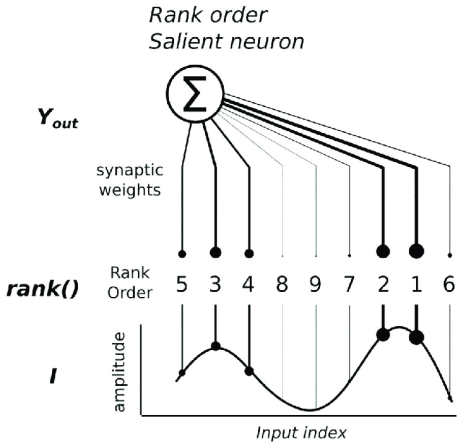



Figure 4.2: Analog signal encoded with ROC. The signal is sampled 9 times over the given time-window which corresponds to 9 input neurons. The samples with the highest values are ordered first, thought to carry the most information.

Furthermore, an important feature that ROC offers is invariance. Changes

in the intensity of the signal or its contrast will not affect the result of its encoded spiketrain. This is a note-worthy advantage over typical rate coding schemes, which may distort the information of the signal during the encoding. The idea of Rank Order Decoding follows Thrope's neuron model where the input's spike order affects the increase of the membrane potential.

$$V_{in} = \sum_{j \in input} mod^{order(a_j)} \boldsymbol{w}_{j,in}$$

In the above equation, $a_j$ denotes the order that the neuron j fired and mod is a chosen number less than 1 that gets raised to $a_j$. In this way, the higher the order (j fires later) the less it affects the neuron's membrane potential. Rank order coding has shown promising results in experiments where it was used to encode either visual [126] or audio signals [127], proving its capability on encoding both temporal and non-temporal data, such as speech and images respectively, in temporal encoded spiketrains.

### 4.2.3 Population Order Coding (POC)

Population Order Coding follows the same principle with ROC, where the information is stored in the order the neurons fire, but each value input value is encoded by a population of neurons and their rank carries the information of the signal. In POC, the order of the neuron population encodes information about a single input value in contrast with ROC, where the order of the spikes encoded information about the input signal in relation to each other. POC gives the ability to encode additional information such as the magnitude of the input signal. It has to be noted that, while ROC input neurons correspond with one input signal each. POC's input population encodes a single input signal. The ordering in POC is done with the following principle. Multiple overlapping receptive fields (expressed by Gaussian-like functions), each corresponding to an input neuron, covers the whole domain of the possible input signal values. This value causes the firing of the input neurons with their firing time being proportional to the value of their corresponding receptive field for the given input [128]. Figure 4.9 shows the spiketrain that encodes the input value shown in the blue line. The order of the spikes emitted is correlated with the membership of the input signal to each receptive field.

It has been shown that biological systems, such as clusters of neurons in mammals' brain, encode information using Population of neurons. A lot

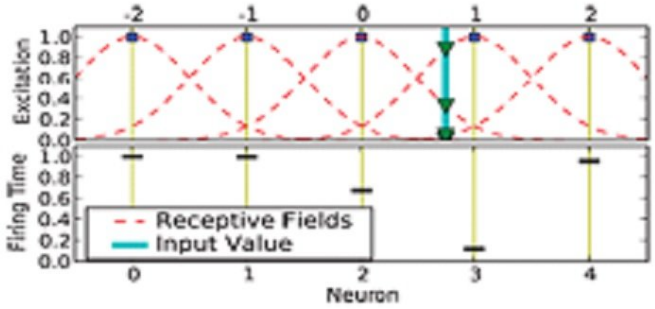



Figure 4.3: Input value encoded with Population Order Coding from a population of 5 neurons with Gaussian receptive fields. The timing of the firing of each neuron is shown in the second diagram where it is shown that the time of spikes emission is proportional to the Gaussian value of each field.

of research work has shown that POC is used in rats (in rat somatosensory cortex [129]), human brains (in auditory system [130], where the authors studied the POC from their experimental results) and in sound encoding in general [131], while it has been also used in artificial SNNs, such as for the development of evolving SNNs [132] and for temporal classification [133].

### 4.2.4 Threshold-based encoding (or Temporal Contrast)

One simple method of temporal encoding is Temporal Contrast, where changes of the input signal that exceed a threshold are denoted as spikes. However, this encoding algorithm requires the acceptance of negative spikes. This is needed, as the input signal may raise through time at first, where it will be encoded as positive spikes, but goes back to lower values afterwards. This fall should be encoded as it may exceed the threshold but has to be denoted differently from the positive spike, most commonly as a negative one (or activation of a different neuron) as shown in figure 4.10. The spiketrain tries to follow the input signal mimicking its behavior but quantifying the possible values. This algorithm is similar to the delta modulation [134], a widely known analog-to-digital converter with whom it carries similar advantages and disadvantages. The only parameters estimation in this encoding algorithm are the value of the threshold and the dt of two consecutive discrete timesteps. It is usually considered that dt = 0 and that the encoded spike-

train does not lose information due to this delay, leaving the only parameter the threshold value. The threshold value in Temporal Contrast is calculated taking into account the whole input signal and is related to its first derivative, so as to optimize the number of spikes in the encoded spiketrain. This encoding algorithm, along with others that will be presented next, is further studied in this paper [135].

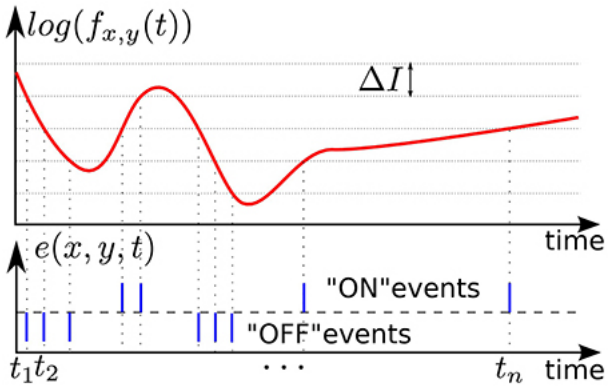



Figure 4.4: Signal through time encoded with Temporal Contrast. Over exceeding the threshold $\Delta I$ causes a positive spike while under exceeding it causes a negative one.

### 4.2.5 Further Reading

In order to overcome the need of the whole signal as it has been stated in the Temporal Contrast algorithm, two more encoding algorithms have been developed, Step-Forward encoding and Moving Window. These 2 encoding schemes follow the same principle as temporal contrast since they both encode information in the difference of the signal through time. Step-Forward encoding has a threshold parameter Th that is set as an input from the engineer and does not use any information of the signal. It calculates if the value of the signal has exceeded a baseline B(t) $\pm$ Th. The baseline is set at the start as the initial value of the signal and the is updated as B(t+1) = B(t) $\pm$ Th depending on the behavior of the signal (plus if the signal over exceeds, minus if it under exceeds and B(t) = B(t+1) if it does not exceed the threshold). High threshold values keep the cause less dense spiketrains while low ones cause spiketrain consisting mostly of emitted spikes, either positive

or negative ones, making an overloaded signal. Moving Window encoding stands between Step-Forward and Temporal Contrast. It also has a user-set threshold Th and a baseline B, but the baseline is calculated as the mean of the previous values of the signal over a time window set by the user, thus creating a moving average filter. Decoding the Step-Forward and Moving Window spiketrains is the same as decoding a Temporal Contrast spiketrain, where for each positive spike the reconstructed signal is added the threshold value and for each negative one decreases by the threshold.

Another approach of coding signals into spiketrains is the use of Finite Impulse Response filters (FIR filters). In Hough Spiker Algorithm (HSA) [136], a filter $h(t)$ of size M time steps is set and the following equation is satisfied:

$$o(t) = (x \circledast h) = \sum_{k=0}^{M} x(t-k)h(k) \tag{4.5}$$

In the above convolution, $o(t)$ is the analog signal and $x(t)$ is the spiketrain. This equation represents the decoding of the signal given the spiketrain. In order for the encoding to be done, the reverse procedure has to be done. This is done by comparing the filter values with the ones of the signal and emitting a spike and substructing the filter from the signal if all the filter values are equal or less than the signal, moving one timestep forward and repeating. Ben's Spike Algorithm (BSA) [137] uses the same idea of convolving the signal with an FIR filter h(t). It calculates two errors E1 and E2 as follows:

$$E1 = \sum_{k=0}^{M} abs(s(t+k) - h(k)) \qquad E2 = \sum_{k=0}^{M} abs(s(t+k)) \tag{4.6}$$

In eq.4.16 the $abs()$ indicates the absolute function and $s(t)$ the input signal. For each timestep t, if E1 is smaller than E2 (plus a threshold) then a spike is emitted and the filter is substructed from the input signal else nothing happens. BSA and HSA algorithms provide the great advantage of not needing negative spikes. The resulting encoded spiketrain contains only positive emitted spikes being more biologically plausible and may be easier to handle in an SNN.

These encoding algorithms are both further studied at [135] (with the exception of HSA). Moreover, Julien Dupeyroux in his work [138], driven by the

potential shown in the use of Spiking Neural Networks in robotics, has developed a library in ROS (Robotic Operating System), in which he implemented the algorithms mentioned before to encode information into spikes in robotic systems. In figure 4.10, Ben's Spike Algorithm (BSA), Temporal Contrast (or Threshold-Based representation - TBR), Step-Forward (SF), and Moving Window (MW) algorithms are compared to each other in features important to the encoded signal.

| **Methods** | **BSA** | **TBR** | **SF** | **MW** |
|---|---|---|---|---|
| **spike train polarity** | + | +/- | +/- | +/- |
| **false encoding at start/end** | yes, both | no | no | yes, at start |
| **cuts into frequency band** | yes | no | no | yes |
| **suppresses (white) noise** | yes, greatly | yes, little | yes, little | yes, greatly |
| **optimization** | non-trivial | non-trivial | easy | non-trivial |

Figure 4.5: BSA, TBR, SF and MW algorithms and the features they grant to the resultant spiketrain.

The scientific field of information coding in spikes offers a lot more coding schemes and algorithms not mentioned here. Since the choice of input coding scheme and algorithm affects the performance of the Neural Network, the developer has to consider which one of the algorithms suits him best, and so is provided with the following papers [139], [140] for further reading.

# Chapter 5

# Learning Methods for Spiking Neural Networks

Having previously described methods of modeling neurons, this section analyzes learning rules that are used in Machine Learning with Spiking Neural Networks.

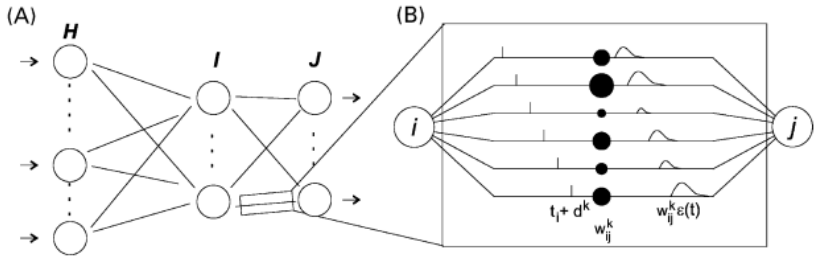



Figure 5.1: (A) Feedforward Spiking Neural Network, (B) A connection between neurons consisting of multiple synaptic terminals

## 5.1 SpikeProp

SpikeProp, a backpropagation-based method for supervised learning, was one of the first methods used to train Spiking Neural Networks [120]. The timing of single-neuron spikes encodes the information during training. The paper's authors also empirically demonstrate that networks of biologically

plausible spiking neurons can perform complex non-linear classification in a fast temporal encoding just as well as rate-coded networks.

### 5.1.1 Network Architecture

The architecture consists of a feedforward network of spiking neurons,in place of artificial neurons, followed by delayed synaptic terminals,as described in [141], see Fig. 5.1. Spiking neurons usually generate action potentials(spikes) when the membrane potential crosses a threshold . The authors consider the membrane potential as an internal neuron state variable. The relationship between spikes and the neuron's internal state variable can be described by any neuron model, but in their implementation of SpikeProp, the Spike Response Model is used, as described in Section .. .

A neuron $j$ in the network, receives input spikes from a set of pre-synaptic neurons $\Gamma_\mathrm{j}$ with firing times $t_\mathrm{j}, i \in \Gamma_\mathrm{j}$ . The dynamics of the neuron's internal state variable $x_\mathrm{j}$ is influenced by the pre-synaptic neurons according to a spike response function $\varepsilon(t)$ and a synaptic weight $w_\mathrm{ij}$.

$$x_\mathrm{j}(t) = \sum_{i \in \Gamma_\mathrm{j}} w_\mathrm{ij} \varepsilon(t - t_\mathrm{i}) \tag{5.1}$$

The response function is responsible for the post synaptic potential,the spike that is being produced and the synaptic weight modulates the height of this potential. Each connection, Fig 5.1 (B), can be broken down further into $m$ fixed individual synaptic terminals,where each synaptic terminal has its own weight and delay . This delay is defined as the difference between the firing time of the pre-synaptic neuron, and the time the post-synaptic potential starts rising, Fig 5.2 (A) . A pre-synaptic spike at a synaptic terminal $k$ is defined as a Post Synaptic Potential of standard height with delay $d^k$ . The unweighted term of a single synaptic terminal that contributes to state variable can be defined as:

$$y^k{}_\mathrm{i}(t) = \varepsilon(t - t_\mathrm{i} - d^k) \tag{5.2}$$

The spike response function $\varepsilon(t)$ has an initial value of zero.The time $t_\mathrm{i}$ is the firing time of pre-synaptic neuron $i$ . The spike response function is given by:

$$\varepsilon(t) = \frac{t}{\tau} e^{1 - \frac{t}{\tau}} \tag{5.3}$$

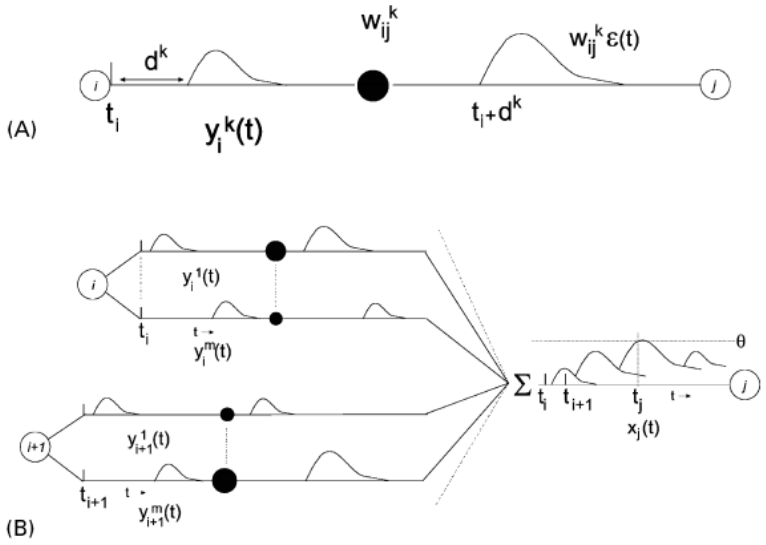



Figure 5.2: Feedforward Spiking Neural Network Connections: (A) single synaptic terminal: the delayed pre-synaptic potential is weighted and produces a post-synaptic potential, (B) two multi-synapse connections with the weighted input being summed at the proceeding neuron.

$\tau$ models the membrane potential decay time constant that determines the rise and decay time of the PSP. The weighted sum of the pre-synaptic contributions is now:

$$\sum_{k=1}^{m} {w_{\mathrm{ij}}}^{k} {y_{\mathrm{i}}}^{k}(t)$$

The variable ${w_{\mathrm{ij}}}^{k}y$) denotes the associated weight of synaptic terminal k ,Fig 5.2 . The firing time $t_{\mathrm{j}}$ of neuron $j$ is equal to the first time when the state variable crosses the threshold *theta* . The threshold is a constant value and remains equal among all neurons.

### 5.1.2 BackPropagation with SpikeProp

The network has three layers as seen in Fig 5.1, H(input),I(hidden),J(output) although more hidden layers can be used. The goal of this error-backpropagating algorithm is to learn a set of firing times ,${t_{\mathrm{j}}}^{d}$, at the output neurons, $j \in J$ for a given set of input patterns $P[t_1...t_{\mathrm{h}}]$. The set of input patterns defines a single input pattern described by single spike times for each neuron $h \in H$. The error-function used in the researchers experiments was the least mean squares error-function, but again other choices can be made for it. Given desired spike times ${t_{\mathrm{j}}}^{d}$ and actual firing times ${t_{\mathrm{j}}}^{a}$ , the error function is defined as:

$$E = 1/2 \sum_{j \in J} ({t_{\mathrm{j}}}^{a} - {t_{\mathrm{j}}}^{d})^2 \tag{5.4}$$

Each synaptic terminal $k$ is considered as a separate connection with weight ${w_{\mathrm{ij}}}^{k}$). The backpropagation equations for the output neurons are:

$$\Delta {w_{\mathrm{ij}}}^{k} = -\eta {y_{\mathrm{i}}}^{k}({t_{\mathrm{j}}}^{a})\delta_{\mathrm{j}} \tag{5.5}$$

$$\delta_{\mathrm{j}} = \frac{{t_{\mathrm{j}}}^{d} - {t_{\mathrm{j}}}^{a}}{\sum_{i \in \Gamma_{\mathrm{j}}} \sum_{l} {w_{\mathrm{ij}}}^{l} (\partial {y_{\mathrm{i}}}^{l}({t_{\mathrm{j}}}^{a}) / \partial {t_{\mathrm{j}}}^{a})} \tag{5.6}$$

Now, the equations for the hidden layers must be defined.

$$\Delta {w_{\mathrm{hi}}}^{k} = -\eta {y_{\mathrm{h}}}^{k}({t_{\mathrm{i}}}^{a})\delta_{\mathrm{i}} \tag{5.7}$$

$$\delta_{\mathrm{i}} = \frac{\sum_{j \in \Gamma^{i}} \delta_{\mathrm{j}} \sum_{k} {w_{\mathrm{ij}}}^{k} (\partial {y_{\mathrm{i}}}^{k}({t_{\mathrm{j}}}^{a}) / \partial {t_{\mathrm{i}}}^{a})}{\sum_{h \in \Gamma_{\mathrm{i}}} \sum_{l} {w_{\mathrm{hi}}}^{l} (\partial {y_{\mathrm{h}}}^{l}({t_{\mathrm{i}}}^{a}) / \partial {t_{\mathrm{i}}}^{a})} \tag{5.8}$$

Output classification was encoded in a way that the neuron coding for the respective class was assigned an early firing time.

### 5.1.3 Important Remarks and Limitations

During experimentation according to the paper, the algorithm would not converge if both negative and positive weights existed. Additionally, for learning convergence it was necessary to incorporate both excitatory and inhibitory neurons.For encoding the input, a method for encoding input variables into temporal spike-time patterns by population coding is used.However,the encoding of input data is depended upon the experimenter's choice and is not restricted to one particular choice by the algorithm so performance can vary . Even though the algorithm looks promising ,it requires the same number of iterations compared to a standard MLP .However, given the explicit use of the time domain for calculations, it is supported that a network of spiking neurons is intrinsically more suited for learning and evaluating temporal patterns than sigmoidal networks.

## 5.2 Spike-Time Dependent Plasticity (STDP)

Spike-Time Dependent Plasticity (STDP) is a biological process, a type of plasticity, that adjusts the strength of neural connections in the brain as described in 2.3.4. More specifically, STDP can be defined as the process that modifies the strength of the connections based on the relative timing of a neuron's output and input action potentials (or spikes). From a mathematician's point of view it can be seen as a temporally asymmetric form of Hebbian learning dependent on neurons' temporal corellations induced by tight temporal correlations between the spikes of pre- and postsynaptic neurons [142] .

STDP with Spiking Neural Networks is a form of unsupervised learning in Machine Learning Terms. During learning, the weight change $\Delta w_{\mathrm{j}}$ of a synapse from a presynaptic neuron $j$ depends on the relative timing between presynaptic spike inputs and postsynaptic spikes. Set of presynaptic spike arrival times at synapse $j$,with count number $f$ is named ${t_{\mathrm{j}}}^{f}$ . Similarly, firing times of postsynaptic neuron are named ${t_{\mathrm{i}}}^{n}$. The total weight change then [33] is defined as:

$$\Delta w_{\mathrm{j}} = \sum_{f=1}^{N} \sum_{n=1}^{N} W({t_{\mathrm{i}}}^{n} - {t_{\mathrm{j}}}^{f}) \tag{5.9}$$

$W(x)$ denotes the learning window,a usual choice is the following:

$$W(x) = \begin{cases} A_+ exp(-x/\tau_+), & \text{for } x>0 \\ -A\_ exp(x/\tau\_), & \text{for } x<0 \end{cases}$$

The values $A_+$ and -A_ may depend on the current value of W(x) and time constants $\tau$ are on the order of 10ms. STDP has been shown to learn "early spike patterns" when a neuron is repeatedly presented with discrete volleys of input spikes, concentrating synaptic weights on inputs that consistently fire early, with the result that the postsynaptic spike latency decreases until it reaches a minimal and stable value. These findings hold true under a continuous regime in which inputs fire at a constant population rate. As a result, STDP can handle a difficult computing problem: localizing a recurring spatiotemporal spike pattern contained in equally dense 'distractor' spike trains [143]. STDP thus enables some form of temporal coding, even in the absence of an explicit time reference. It has been reported that repeating spatio-temporal spike patterns with millisecond precision exist in electrophysiological experiments [144]. In [143] they show how STDP can learn the difficult task of detecting these repeating patterns, demonstrating once again how spiking neural networks can solve difficult tasks of this type with the appropriate learning rules.

### 5.2.1 STDP detecting repeating spike train patterns

One example of such a repeating pattern is shown in Fig 5.3 .The problem is complicated by the fact that neither the population firing rate nor the firing rates of the neurons participating in the pattern are unique during the periods when the pattern is present. This type of situation requires taking the spike times into account. Researchers show how a neuron making use of STDP can solve this problem leveraging the fact that a pattern is a series of spike coincidences. STDP is known to have the effect of concentrating synaptic weights on repeated early firing inputs, resulting in a decrease in postsynaptic spike latency when a neuron is repeatedly presented with similar volleys of input spikes.In other words,these inputs systematically lead to a shaping of the neuron's selectivity. This shaping was achieved in a variety of background noise conditions, as well as in situations where spiking latencies and firing rates, or synchronization, gave contradictory information [145] [146] [147] . The researchers indicate the limitation of these studies requiring an explicit

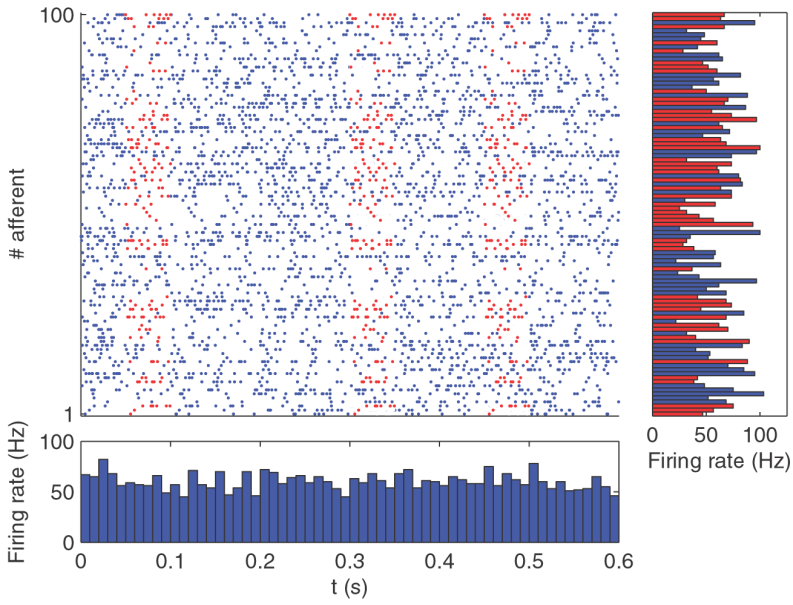



Figure 5.3: In red a repeating 50 ms long pattern that concerns 50 afferents among 100 is indicated. The bottom panel plots the population-averaged firing rates over 10 ms time bins. The right panel plots the individual firing rates averaged over the whole period [143].

time reference and wonder if STDP can recognize the repeating pattern in the absence of a time reference.

To test this an arbitrary pattern was inserted as in Fig 5.3, and investigated whether a single receiving STDP was able to learn it in an unsupervised manner. Input spikes were simulated according to a Poisson process where neurons fire stochastically and independently. The arbitrary pattern was purposely hidden from the firing rate of the neurons so it would be impossible to solve using the firing rates alone. By reinforcing the synaptic connections with the afferents that took part in firing the neuron(shaping the neuron's selectivity). When the pattern is shown again, the likelihood that the neuron will fire again is increased . Apart from shaping the neuron's selectivity, it also allows for convergence by saturation when all the spikes in the pattern that precede the postsynaptic spike already correspond to maximally potentiated synapses, and all are necessary to reach the threshold. Spikes outside the pattern are depressed so they don't contribute to the membrane potential. This leads to an absence of false "alarms" meaning that after learning the neuron only spikes when the pattern is present (Fig 5.4). If this occurs in the brain ,information about a stimulus can be readily available since neurons will fire on stimulus onset as postulated in [148].

### 5.2.2 STDP and oscillations

STDP also allows to incorporate oscillations as it was shown to be able to select only phase-locked inputs among a broad population with random phases, turning the postsynaptic neuron into a coincidence detector. By teaching the network to respond to certain phase-locked inputs it can coordinate and synchronize neural activity [33].One example demonstrated in the mentioned study, is how an owl is able to coordinate the neural activity between its two ears fast enough resulting in a reaction time before turning its head, of about 100ms. Learning with STDP selects synapse connections in such a way for the spikes to arrive coherently . Another example of such need for coordination with oscillations is during saccadic eye movement. To effectively make use of the spatial information contained in the luminance modulations resulting from eye movements, analysis of neural activity needs to be timed relative to the occurrence of saccades. For example, a spike from the same neuron carries a different informational value if occurring during early fixation, when the input change caused by the preceding saccade still exerts its influence, or later, when temporal changes are only caused by ocular drift [23],see fig

5.5.The frequencies of visual cortical oscillations can be controlled locally and can be modulated by and locked to external stimuli [149].

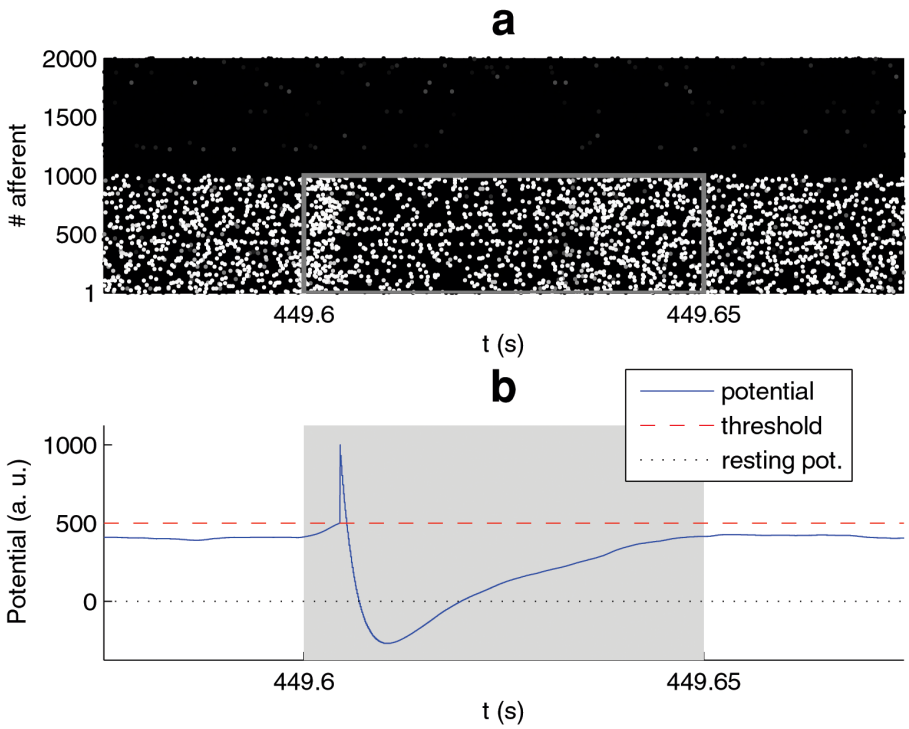



Figure 5.4: (a)The pattern is represented as grey line rectangle. Take note of the cluster of white spikes at the beginning: Most of the synapses that correspond to the pattern's early spikes have been potentiated by STDP. It's worth noting that almost all synaptic connections with afferents that aren't engaged in the pattern have been fully suppressed. (b) Over the same range as above, the membrane potential is displayed as a function of time. The abrupt spike that corresponds to the above-mentioned cluster is readily visible [143].

## 5.3 Surrogate Gradient Learning in Spiking Neural Networks

We use this paper to introduce you to the surrogate gradient methods, as surrogate gradients are used widely in other learning methods which we will

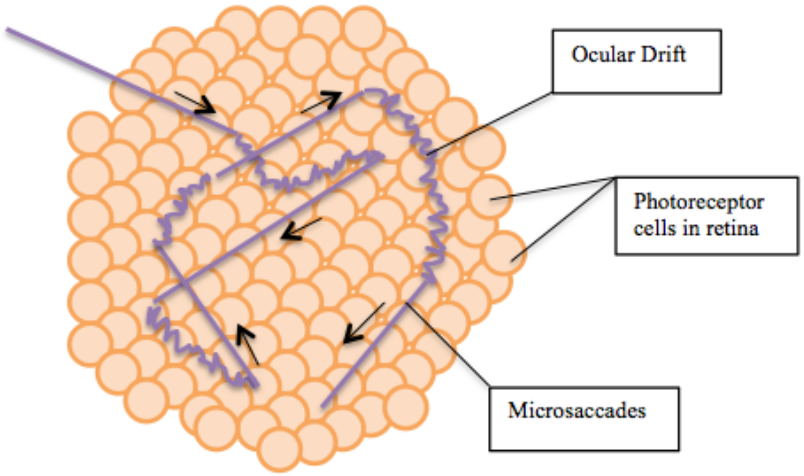



Figure 5.5: Saccadic movements and ocular drift

see later. Additionally the authors formally map SNNs to RNNs. Formulating SNNs as RNNs allows to transfer and apply existing training methods for RNNs [150]. This mapping is important if we want to train a larger number of layers with Spikes .This paper also introduces how to solve the credit assignment problem with many layers and also the major issues with multi-layer Spiking Neural Networks that make training much harder than traditional neural networks.

### 5.3.1 Mapping RNNs to SNNs

In the broadest sense, RNNs are networks whose state evolves over time according to a set of recurrent dynamical equations. Such dynamical recurrence can occur by the presence of recurrent synaptic connections between neurons in the network. In today's machine learning research this is the most common definition of what an RNN is. This occurs, for example, when stateful neuron or synapse models with internal dynamics are employed. These dynamics are intrinsically recurrent since the network's state at a given time step recurrently depends on its state at prior time steps. The paper as per usual

uses the LIF neuron model(equation 5.30 for the membrane voltage) which we have seen in previous sections.However, an additional term for recurrent connections in the synaptic current equation can be added:

$$\frac{\mathrm{d}I_i^{(l)}}{\mathrm{d}t} = -\underbrace{\frac{I_i^{(l)}(t)}{\tau_{\text{syn}}}}_{\text{exp. decay}} + \underbrace{\sum_j W_{ij}^{(l)} S_j^{(l-1)}(t)}_{\text{feed-forward}} + \underbrace{\sum_j V_{ij}^{(l)} S_j^{(l)}(t)}_{\text{recurrent}} \tag{5.10}$$

the sum runs over all presynaptic neurons $j$ and $W_{ij}^{(l)}$ are the corresponding afferent weights from the layer below. The $V_{ij}^{(l)}$ correspond to explicit recurrent connections within each layer,this corresponds to additional term we are talking about. The authors also state that with recurrent connections a single LIF neuron can be simulated with two linear differential equations whose initial conditions change instantly anytime a spike occurs.So , the reset term can be inserted in the membrane potential equation as an extra term that instantaneously decreases the membrane potential by $(\vartheta - U_{\text{rest}})$ whenever the neuron emits a spike:

$$\frac{\mathrm{d}U_i^{(l)}}{\mathrm{d}t} = -\frac{1}{\tau_{\text{mem}}}\left(\left(U_i^{(l)} - U_{\text{rest}}\right) + RI_i^{(l)}\right) + S_i^{(l)}(t)\left(U_{\text{rest}} - \vartheta\right) \tag{5.11}$$

The above equations need to be approximated in discrete for a computer simulation .Also,the output spike train $S_i^{(l)}[n]$ of neuron $i$ in layer $l$ at time step $n$ needs to expressed as a nonlinear function of the membrane voltage $S_i^{(l)}[n] \equiv \Theta\left(U_i^{(l)}[n] - \vartheta\right)$ where $\Theta$ denotes the Heaviside step function and $\vartheta$ corresponds to the firing threshold. The simulation time step also needs to be small for the approximation to work properly.Equation 5.10 becomes:

$$I_i^{(l)}[n+1] = \alpha I_i^{(l)}[n] + \sum_j W_{ij}^{(l)} S_j^{(l)}[n] + \sum_j V_{ij}^{(l)} S_j^{(l)}[n] \tag{5.12}$$

Decay strength: $\alpha \equiv \exp\left(-\frac{\Delta_t}{\tau_{\text{syn}}}\right)$. For finite and positive $\tau_{\text{syn}}$:$0 < \alpha < 1$. Also, $S_j^{(l)}[n] \in \{0, 1\}$. The variable $n$ is used to denote the time step. The equation 5.11 is now:

$$U_i^{(l)}[n+1] = \beta U_i^{(l)}[n] + I_i^{(l)}[n] - S_i^{(l)}[n] \tag{5.13}$$

with $\beta \equiv \exp\left(-\frac{\Delta_t}{\tau_{\text{mem}}}\right)$ With these two equations at hand the state of neuron $i$ can be found by the instantaneous synaptic currents $I_i$ and the membrane

voltage $U_i$ (Box. 1]. The computations necessary to update the cell state can be unrolled in time,according to the authors of the paper, as shown in the computational graph in fig 5.6. This is a usual illustration method used for

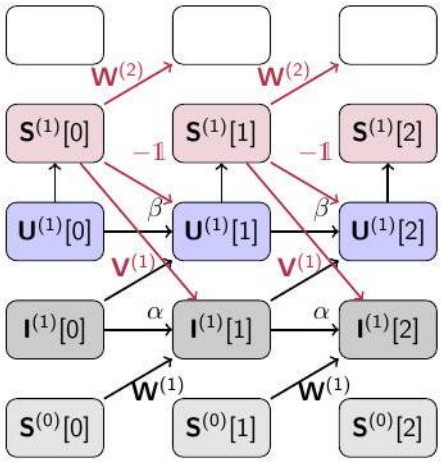



Figure 5.6

Recurrent Neural Networks so the mapping of SNNs into RNNs comes very handy here.Time steps flow from left to right.

- Input spikes $\mathbf{S}^{(0)}$ are inserted in the bottom and propagate upwards to higher layers.
- The synaptic currents $\mathbf{I}$ are decayed by $\alpha$ in each time step and fed into the membrane potentials $\mathbf{U}$.The $\mathbf{U}$ are decaying over time as characterized by $\beta$.
- Spike trains $S$ are generated by applying a threshold nonlinearity to the membrane potentials U in each time step.
- Spikes causally affect the network state (red connections).

The authors of the paper also inform us about how the spikes can be communicated inside the network First, each spike causes the membrane potential of the neuron that emits the spike to be reset. Second, each spike may be fed back to the same neuronal population via recurrent connections $\mathbf{V}^{(1)}$. It can also be fed $\mathbf{W}^{(2)}$ to a network layer further down or, fed to a readout layer on which a cost function is defined.

### 5.3.2 Credit Assignment Problem

As in every learning rule the first step is to choose a cost/loss function which gets reduced when the network starts learning what we want it to learn. The second step is to choose how the weights of the connections update during training to reduce the cost/loss function. This is called the credit assignment problem. One way to solve this problem is to use spatial credit assignment by backpropagation which comes at significant memory costs as the gradients need to be communicated back to network since we are storing all neuron states . Let's see how backpropagation works in Recurrent Neural Networks: Backpropagation in RNNs can be applied by "unrolling": an auxiliary network is created by making copies of the network for each time step.he same rule can be applied to RNNs. In this case the recurrence is "unrolled" (Fig 5.7 ) meaning that an auxiliary network is created by making copies of the network for each time step.

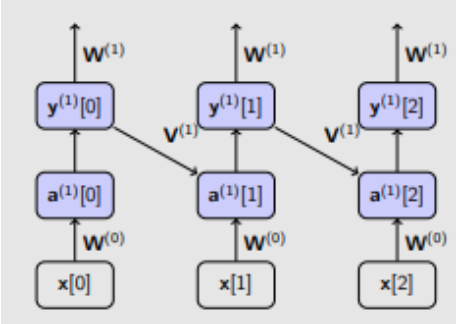



Figure 5.7

The unrolled network is simply a deep network with shared feedforward weights $\mathbf{W}^{(l)}$ and recurrent weights $\mathbf{V}^{(l)}$, on which the standard BP applies:

$$\Delta W_{ij}^{(l)} \propto \frac{\partial}{\partial W_{ij}^{(l)}} \mathcal{L}[n] = \sum_{m=0}^{t} \delta_i^{(l)}[m] y_j^{(l-1)}[m] \tag{5.14}$$

$$\Delta V_{ij}^{(l)} \propto \frac{\partial}{\partial V_{ij}^{(l)}} \mathcal{L}[n] = \sum_{m=1}^{t} \delta_i^{(l)}[s] y_j^{(l)}[m-1] \tag{5.15}$$

$$\delta_i^{(l)}[n] = \sigma'\left(a_i^{(l)}[n]\right) \left( \sum_k \delta_k^{(l+1)}[n] W_{ik}^{\top,l} + \sum_k \delta_k^{(l)}[n+1] V_{ik}^{\top,l} \right) \tag{5.16}$$

The superscripts $l = 0, \ldots, L$ denote the layer (0 is input, $L$ is output) .With $\alpha_i^{(l)}[n] = \sum_j W_{ij} x_j$ we denote the total input to the neuron i, $y_j$ is the output of neuron j, and $\eta$ a small learning rate.Also, $\sigma'$ is the derivative of the activation function, and $\delta_i^{(l)}$ is the error of output neuron $i$ and $T$ indicates the transpose. Applying BP to an unrolled network is called backpropagation through time (BPTT).

The second way to solve this is to use a temporal credit assignment (Fig 5.6 is considered a temporal credit assignment problem) . This type of temporal assignment can be further subdivided in the backward and forward methods.

1. Backward Method: We can use BPTT as in the spatial credit assignment, we propagate the errors backwards through time after a forward pass.

2. Forward Method: All the necessary information to compute the gradients is propagate forwards in time [151] . For example, the "forward gradient" of the feed-forward weight $\mathbf{W}$ becomes:

$$\Delta W_{ij}^m \propto \frac{\partial \mathcal{L}[n]}{\partial W_{ij}^m} = \sum_k \frac{\partial \mathcal{L}[n]}{\partial y_k^{(L)}[n]} P_{ijk}^{L,m}[n] P_{ijk}^{(l,m)}[n] = \frac{\partial}{\partial W_{ij}^m} y_k^{(l)}[n]$$

$$P_{ijk}^{(l,m)}[n] = \sigma'\left(a_k^{(l)}[n]\right)\left(\sum_{j'} V_{ij'}^{(l)} P_{ijj'}^{(l,m)}[n-1] + \sum_{j'} W_{ij'}^{(l)} P_{ijj'}^{(l-1,m)}[n-1] + \delta_{lm} y_i^{(l-1)}[n-1]\right)$$

The gradients with respect to recurrent weights $V_{ij}^{(l)}$ can similarly be computed. The backward optimization method is more efficient in terms of computation, but requires maintaining all the inputs and activations for each time step. So, space complexity for each layer is $O(NT)$, where $N$ is the number of neurons per layer and $T$ is the number of time steps. The forward method requires maintaining variables $P_{ijk}^{(l,m)}$, resulting in a $O\left(N^3\right)$ space complexity per layer. However, if simplifications are applied this complexity can be brought down to O(N), such as in the Decolle learning rule which we describe later. The simplifications can also reduce the computational complexity. However the above algorithms cannot be directly applied . One issue is the non-deferentially of the spiking nonlinearity. The derivative of the neural activation function $\sigma' \equiv \frac{\partial y_i^{(l)}}{\partial a_i^{(l)}}$ is a problem because for a spiking neuron, we have $S(U(t)) = \Theta(U(t) - \vartheta)$, whose derivative is zero everywhere

except at $U = \vartheta$, where it is ill defined. Standard BP, apart from the expensive computation,memory communication requirements, it is poorly suited for neuromorphic hardware nor is it biologically realistic. This type of hardware has locality requirements that prohibits BP all together. The forward method might be more applicable however the scaling of the above methods is not suitable for many SNN models according to the authors. Solving the first has many approaches:

1. Using biologically inspired local learning rules for the hidden units
2. Translating conventionally trained "rate-based" neural networks to SNNs
3. Smoothing the network model to be continuously differential
4. Defining a surrogate gradient (SG) as a continuous relaxation

The main contribution of this research paper is for the last approach, using surrogate gradients and also inform us about smoothing methods too. However , we are only interested in defining the surrogate gradient approach.

### 5.3.3 Surrogate Gradients Approaches

The surrogate gradient approach can be further broken down into two approaches:

1. SGs that make up a continuous relaxation of the non-smooth spiking nonlinearity. This does not influence the optimization algorithm(Fig.5.8)
2. SGs that affect locality of the underlying optimization algorithms themselves to improve the computational and/or memory access overhead of the learning process.

One major advantage with Surrogate Gradients we don't need to specify which coding method is to be used in the hidden layers. In the first surrogate gradient approach researchers replace the derivative of the spiking nonlinearity with the derivative of a smooth function. This approach is used in Decolle and quite simple to implement and handy as it can be combined with auto-differentiation tools. The superspike algorithm uses a three factor online learning rule using a fast sigmoid to construct a SG. Surrogate Gradients are also used in e-Prop which we will examine after Decolle.

### 5.3.4 Surrogate Gradient Issues

As the use of SGs to train SNNs progresses to deeper architectures, more issues, similar to those seen in ANNs, are likely to occur. SGs formed from sigmoidal activation functions,that have vanishing gradient issues. Another issue is the potential bias that SGs introduce into the learning dynamics.

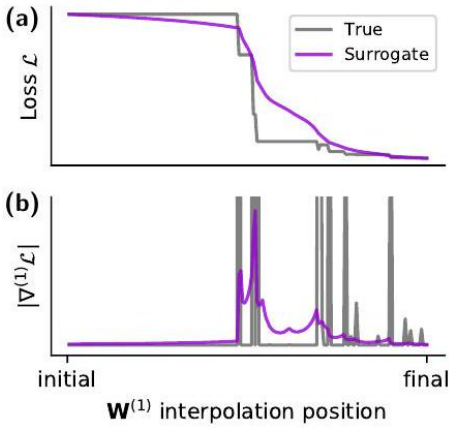



Figure 5.8: For more details refer to the original Surrogate Gradient paper [150]. What's important here is to look how the surrogate gradient approach allows us to have continuous function

### 5.3.5 Conclusion

In this chapter we introduced the reader to the major issues in training spiking neural networks and how these hinder SNNs to reach the capacity of ANNs. The mapping to RNNs allows SNNs to be more approachable to machine learning engineers that don't have a neuroscience background and makes the scaling issues easier to understand.This chapter does not introduce new a learning algorithm but it allows us to apply BPTT with the use of surrogate gradients. Surrogate gradients is a very successful approach in solving the non-differentiable gradient issue, as seen from the amount of new SNN algorithms involving them.We included this section as all the following algorithms in our dissertation involve surrogate gradients and because we are going to train SNN networks using BPTT and SGs.

## 5.4 SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks

Superspike is a surrogate gradient approach, a nonlinear voltage-based three factor learning rule capable of training multi-layer networks of deterministic integrate-and-fire neurons to perform nonlinear computations on spatiotemporal spike patterns. By translating Backprop to the spiking domain, it's one of the few initiatives to tackle the difficulty of training SNNs with hidden units to process precisely timed input and output spike trains.

The partial derivatives of this approach are of the form $\partial S_i(t)/\partial w_{ij}$ where $S_i(t) = \sum_k \delta\left(t - t_i^k\right)$ is the spike train of the hidden neuron $i$ and $w_{ij}$ is a hidden weight.

In comparison to other methods in the literature, Superspike allows multilayer networks of deterministic LIF neurons to solve tasks involving spatiotemporal spike pattern transformations without the need for noise injection, even when hidden units are initially completely silent, as described in the SuperSpike research paper. Instead of the postsynaptic spike train, the partial derivative of the hidden unit outputs is approximated as the product of the filtered presynaptic spike train and a nonlinear function of the postsynaptic voltage.

### 5.4.1 SuperSpike learning rule

It is desired that a single LIF neuron emits a given target spike train $\hat{S}_i$ for a given input. This problem can be considered an optimization problem of minimizing the van Rossum distance [152] between $\hat{S}_i$ and the actual output spike train $S_i$. First, let us describe what the van Rossum distance is. The van Rossum distance aims to solve the task of discriminating between two spike trains .M. C. W. van Rossum introduced a measure for the distance between two spike trains. With the goal of a simple distance measure, given a spike train with spike times $t_i$

$$f^{\text{orig}}(t) = \sum_i^M \delta(t - t_i) \tag{5.17}$$

where it is assumed that all $t_i > 0$. The delta function associated with each spike is replaced with an exponential function, that is, an exponential tail is

added to all spikes,

$$f(t) = \sum_{i}^{M} H\left(t - t_i\right) e^{-(t-t_i)/t_c} \tag{5.18}$$

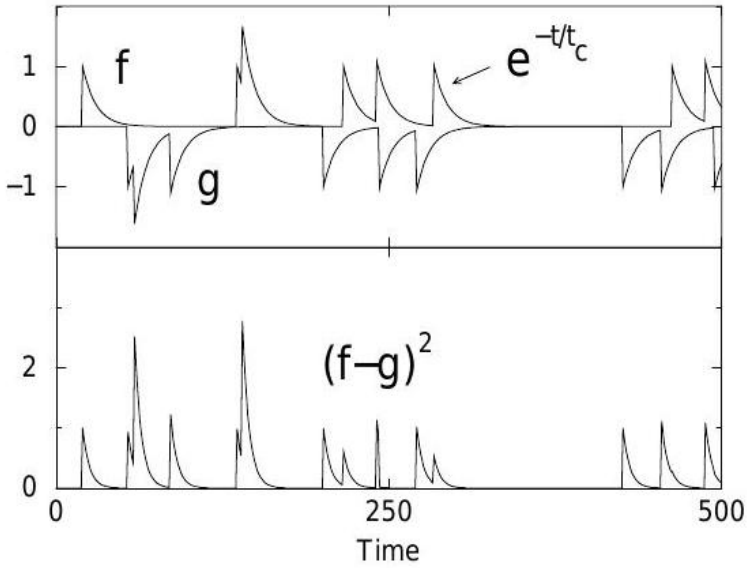



Figure 5.9: (Top) Two spike trains (one flipped) are convolved with an exponential with time constant $t_c$. (Bottom) The difference squared of the spike trains. The integral of the curve gives the desired distance [152]

$t_c$ is the time constant of the exponential function and $H$ is the Heaviside step function ($H(x) = 0$ if $x < 0$ and $H(x) = 1$ if $x \geq 0$). Any function can be used to convolve with other than the exponential but the exponential was chosen because of its biological proximity. The distance between two trains $f$ and $g$ is defined as (see Fig.5.9)

$$D^2(f,g)_{t_c} = \frac{1}{t_c}\int_0^{\infty} [f(t) - g(t)]^2 dt \tag{5.19}$$

The distance is the Euclidean distance of the two filtered spike trains($t_c$ as a free parameter). M. C. W. van Rossum gives us a sense of the distance with the following example.He wants us to consider the two limits of $t_c$. For $t_c$ much smaller than the interspike interval, the smeared functions $f$ and $g$

contribute to the integral only if the spikes are not more than $t_c$ apart. This is reminds us of coincidence detection. For most spike trains, coincident spikes can be neglected in the limit of zero $t_c$; thus, if $f$ contains $M$ and $g$ contains $N$ spikes, one has

$$\lim_{t_c \to 0} D^2(f,g)_{t_c} = \frac{1}{t_c} \int_0^\infty \left[ f^2(t) + g^2(t) \right] dt = \frac{M+N}{2} \tag{5.20}$$

This distance basically counts the noncoincident spikes. However, for large $t_c$, the main contribution to the integral comes from times when the last spike has passed but the exponent has still not decayed.The integral is mostly influenced by times when the last spike has passed but the exponent has not yet decayed. Assuming that $f(g)$ contains $M(N)$ spikes, one can approximate

$$\lim_{t_c \to \infty} D^2(f,g)_{t_c} = \frac{1}{t_c} \int_0^\infty \left( M e^{-t/t_c} - N e^{-t/t_c} \right)^2 dt = \frac{(M-N)^2}{2} \tag{5.21}$$

In this limit, $D$ measures the difference in total spike count.The upper limit of the integral needs to be taken to be infinity as we interested for the tail of the last spike that died out rather than the time of the last spike

The distance thus interpolates between the two extremes of coincidence detection and measuring difference in total spike count. After filtering, the spike trains this leaves a sum of $N+M$ terms. Changing integration variables leads to an alternative expression for the distance,

$$D^2(f,g) = \frac{1}{2} \int_{-\infty}^\infty C_{f-g,f-g}(t) e^{-|t|/t_c} dt \tag{5.22}$$

where $C_{f-g,f-g}(t)$ is the autocorrelation of the difference of the raw spike trains, $f^{\text{orig}}(t) - g^{\text{orig}}(t)$.This demonstrates that the distance can be represented as a weighted integral over the autocorrelation, with the weighting varying with $t_c$. It also demonstrates that the distance is unaffected by time reversal, i.e., the distance would be the same if the exponential tails were attached to the spikes on the opposite sides.Regarding the choice of the exponential M. C. W. van Rossum states that the convolution in a higher-order neuron for short and medium periods can be read as postsynaptic potentials. Longer $t_c$ appears to be better served by slower second messenger or calcium-induced currents. Now lets us continue with optimization problem of minimizing the van Rossum distance [152] between $\hat{S}_i$ and the actual output spike train $S_i$:

$$L = \frac{1}{2} \int_{-\infty}^t ds \left[ \left( \alpha * \hat{S}_i - \alpha * S_i \right)(s) \right]^2 \tag{5.23}$$

where $\alpha$ is a normalized smooth temporal convolution kernel. Because they can be easily computed online and implemented as electrical or chemical traces in neurobiology, double exponential causal kernels are used. Computing the gradient of the above equation with respect to the synaptic weights $w_{ij}$:

$$\frac{\partial L}{\partial w_{ij}} = -\int_{-\infty}^{t} ds \left[\left(\alpha * \hat{S}_i - \alpha * S_i\right)(s)\right] \left(\alpha * \frac{\partial S_i}{\partial w_{ij}}\right)(s) \tag{5.24}$$

in which the derivative $\partial S_i / \partial w_{ij}$ is the derivative of the spike train . This derivative for most neuron models it is zero except at spike times at which it is not defined. This is a huge problem for backpropagation . Previous training algorithms avoid this problem by either performing optimization directly on the membrane potential $U_i$ or by introducing noise which makes the likelihood of the spike train $\langle S_i(t) \rangle$ a smooth function of the membrane potential. Superspike combines these two approaches by replacing the spike train $S_i(t)$ with a continuous auxiliary(helper) function $\sigma\left(U_i(t)\right)$ of the membrane potential. For performance reasons, the authors of Superspike choose $\sigma(U)$ to be the negative side of a fast sigmoid . This "helper" function" replaces the derivative of the spike train as follows:

$$\frac{\partial S_i}{\partial w_{ij}} \quad \rightarrow \quad \sigma'\left(U_i\right) \frac{\partial U_i}{\partial w_{ij}} \tag{5.25}$$

To compute the derivative $\partial U_i / \partial w_{ij}$ in the expression above, for current-based LIF models the membrane potential $U_i(t)$ as a spike response model:

$$U_i(t) = \sum_j w_{ij} \left(\epsilon * S_j(t)\right) + \left(\eta * S_i(t)\right) \tag{5.26}$$

The causal membrane kernel $\epsilon$ corresponds to the postsynaptic potential (PSP) shape and $\eta$ describe the spike dynamics and reset. Due to the second term, $U_i$ depends on its own past through its output spike train $S_i$.

The dependence of its own past does not allow us to compute the derivative $\frac{\partial U_i}{\partial w_{ij}}$ directly. However, by ensuring low firing rates(by adding homeostatic mechanisms that regularize neuronal activity levels) we can ignore the second term.

Ignoring the second term, the equation becomes the filtered presynaptic activity $\frac{\partial U_i}{\partial w_{ij}} \approx \left(\epsilon * S_j(t)\right)$. Biologically, this can be interpreted as the concentration of neurotransmitters at the synapse.

Substituting this approximation back into Eq. (4.26), the gradient descent learning rule for a single neuron becomes:

$$\frac{\partial w_{ij}}{\partial t} = r \int_{-\infty}^{t} ds \quad \underbrace{e_i(s)}_{\text{Error signal}} \quad \underbrace{\alpha * \underbrace{\sigma'\left(U_i(s)\right)}_{\text{Post}} \underbrace{\left(\epsilon * S_j\right)(s)}_{\text{Pre}}}_{\equiv \lambda_{ij}(s)} \tag{5.27}$$

The learning rate is symbolized with $r$ and short notation for the output error signal $e_i(s) \equiv \alpha * \left(\hat{S}_i - S_i\right)$ and the eligibility trace (a temporary record of the occurrence of an event) $\lambda_{ij}$. In practice the expression is evualated on minibatches and Superspike suggests using a per-parameter learning rate $r_{ij}$ to speed up learning.

The last equation corresponds to the SuperSpike learning rule for output neuron $i$. By redefining the error signal $e_i$ as a feedback signal, the same rule for hidden units can be used as well. Important properties:

- It includes Hebbian term which combines pre- and postsynaptic activity
- The learning rule is voltage-based
- is a nonlinear Hebbian rule due to the occurrence of $\sigma'\left(U_i\right)$
- The causal convolution with $\alpha$ acts as an eligibility trace to solve the distal reward problem due to error signals arriving after an error was made [153]
- It is a three factor rule in which the error signal plays the role of a third factor [154] . The error signal is specific to the postsynaptic neuron.

The "distal reward problem" is an explanatory conundrum that occurs when rewards arrive seconds after reward-triggering events. How does the brain recognize which firing patterns of which neurons are responsible for the reward if 1) the patterns are no longer there when the reward arrives and 2) all neurons and synapses are active during the reward's waiting period?

### Neuron model

The LIF neuron model with current-based synaptic input is used to use the integral form. For the membrane dynamics simulation the voltage $U_i$ of

neuron $i$ can be computed as follows:

$$\tau^{\text{mem}} \frac{dU_i}{dt} = \left(U^{\text{rest}} - U_i\right) + I_i^{\text{syn}}(t) \tag{5.28}$$

in which the synaptic input current $I_i^{\text{syn}}(t)$ changes according to the following equation:

$$\frac{d}{dt} I_i^{\text{syn}}(t) = -\frac{I_i^{\text{syn}}(t)}{\tau^{\text{syn}}} + \sum_{j \in \text{pre}} w_{ij} S_j(t) \tag{5.29}$$

The value of $I_i^{\text{syn}}(t)$ jumps by an amount $w_{ij}$ at the moment of spike arrival from presynaptic neurons $S_j(t) = \sum_k \delta\left(t - t_j^k\right)$ . $\delta$ denotes the Dirac $\delta$ - function and $t_j^k (k = 1, 2, \cdots)$ are firing times of neuron $j$. / To see the values of the following constants read the original Superspike paper as we are not that interested in the implementation details here but more in the theoretical basis of the learning method.

- An action potential is triggered when the membrane voltage of neuron $i$ reaches a value above a threshold value $\vartheta$ .
- Following a spike the voltage $U_i$ remains clamped at $U_i^{\text{rest}}$ for $\tau^{\text{ref}}$ to emulate a refractory period.
- After generation, spikes are propagated to other neurons with an axonal delay .

**Plasticity model**

The learning rule can be interpreted as a nonlinear Hebbian three factor rule. The nonlinear Hebbian term detects coincidences between presynaptic activity and postsynaptic depolarization. These spatiotemporal coincidences at the single synapse $w_{ij}$ are then stored transiently by the temporal convolution with the causal kernel $\alpha$ . This step can be interpreted as a synaptic eligibility trace.

All necessary quantities are computed online without the need to propagate error signals backwards through time. So, Superspike can be interpreted as an implementation of real-time recurrent learning (RTRL) [151] for spiking neural networks.

In the model, all the complexity of neural feedback of learning is absorbed into the per-neuron signal $e_i(t)$. Because it's unclear whether and how

such erroneous feedback is sent to individual neurons in biology the authors tested numerous ways which we won't go into much detail. The previously mentioned equation of the Superspike learning rule is integrated over finite temporal intervals before updating the weights. The full learning rule can be written as follows:

$$\Delta w_{ij}^{k} = r_{ij} \int_{t_k}^{t_{k+1}} \underbrace{e_i(s)}_{\text{Error signal}} \quad \alpha * (\underbrace{\sigma'\left(U_i(s)\right)}_{\text{Post}} \underbrace{\left(\epsilon * S_j\right)(s)}_{\text{Pre}}) ds \tag{5.30}$$

The evaluation of the Superspike learning rule equation can be described as follows: i) evaluation of presynaptic traces, ii) evaluation of Hebbian coincidence and computation of synaptic eligibility traces, iii) computation and propagation of error signals, and iv) integration of the given Superspike equation and weight update.

### 5.4.2 Learning Signals

#### Presynaptic traces

Because $\epsilon$ is a double exponential filter, the temporal convolution in the expression of the presynaptic traces (Eq. 5.30), can be evaluated efficiently online by exponential filtering twice. First , the integration the single exponential trace

$$\frac{dz_j}{dt} = -\frac{z_j}{\tau_{\text{rise}}} + S_j(t) \tag{5.31}$$

in every time step which is then fed into a second exponential filter array

$$\tau_{\text{decay}} \frac{d\tilde{z}_j}{dt} = -\tilde{z}_j + z_j \tag{5.32}$$

with $\tilde{z}_j(t) \equiv \left(\epsilon * S_j\right)(t)$ which now implements the effective shape of a PSP in the model.

#### Hebbian coincidence detection

To evaluate the Hebbian term , the surrogate partial derivative $\sigma'\left(U_i\right)$ is evaluated in every time step.

The outer product between the delayed presynaptic traces $\tilde{z}_j(t-\Delta)$ and the surrogate partial derivatives $\sigma'\left(U_i\right)(t-\Delta)$ is calculated in every time step.

### Synaptic eligibility traces

To implement the synaptic eligibility trace as given by the temporal filter $\alpha$, the values of Hebbian product term are filtered with two exponential filters $\mathrm{z_j}$ . These traces now need to be computed for each synapse $w_{ij}$ which makes the algorithm scale as $O\left(n^2\right)$ for $n$ being the number of neurons. For SuperSpike to function properly, it is important that these transients are long enough to coincide with any related error signal $e_i(t)$. The duration of the transient in the model is given by the filter kernel shape used to compute the van Rossum distance.

### Error signals

Output error signals are linked to output units that have a specific target signal. Their details are determined by the underlying chosen cost function .

At the level of output errors two ways can be used to to learn the output.To learn precisely timed output spikes, the output error signals were exactly given by $e_i = \alpha * \left(\hat{S}_i - S_i\right)$ for an output unit $i$. As can be seen from the equation, the error signal $e_i$ is zero only if the target and the output spike train exactly match with the temporal precision of our simulation. All cost function values were computed online as the root mean square in the superspike paper.

The second way is to classify input spike patterns , we introduced some slack into the computation of the error signal. Instantaneous negative error feedback is given by $e_i = -\alpha * S_i^{\mathrm{err}}$ for each unnecessary additional spike $S_i^{\mathrm{err}}$ . A positive feedback signal $e_i = \alpha * S_i^{\mathrm{miss}}$ when a stimulus failed to emit an output spike when it should have .

### Feedback signals

Feedback signals, are derived from output error signals by feeding them back to the hidden units.

Different credit assignment strategies for hidden units can be used. Symmetric, random and uniform feedback are some examples of feedback signals.

Symmetric feedback:computed as the weighted sum $e_i = \sum_k w_{ki} e_k$ of the downstream error signals using the actual feed-forward weights $w_{ik}$.

Random feedback: They are computed as the random projection $e_i = \sum_k b_{ki} e_k$ with random coefficients $b_{ki}$ drawn from a normal distribution with

zero mean and unit variance,

Uniform feedback: all weighting coefficients were simply set to one $e_i = \sum_k e_k$ corresponding closest to a single global third factor distributed to all neurons, (a diffuse neuromodulatory signal). For the weight updates implementation read section 4.4.2 of the Superspike paper.

### 5.4.3 Multi-Layer Training and Limitations

The form 5.30 requires an extension to hidden layers. The same learning rule for hidden units can be used, with the modification that that $e_i(t)$ becomes a complicated function which depends on the weights and future activity of all downstream neurons. This non-locality in space and time presents serious problems, both in terms of biological plausibility and technical feasibility. Technically, this computation requires either backpropagation through time through the PSP kernel or the computation of all relevant quantities online as in the case of RTRL.

### 5.4.4 Conclusion

The authors suggest that instead of only one global feedback signal, a higher dimensional neuromodulatory or electrical feedback signal for learning potentially with some knowledge of the feedforward pathway is needed. We suggest that oscillations can serve this purpose of intelligent neuromodulation .The brain employs a method of compartmentalizing calculations, which may allow it to perform more complex and difficult tasks, but it also necessitates the need of central control to integrate data from various brain areas. Long-range neuromodulator projection divergence appears to be well-suited to coordinating communication between brain areas. Neural transmission is characterized by oscillatory brain activity [155]. Thus, neuromodulators' ability to modify signal transmission in a frequency-dependent way provides a deeper degree of control.

## 5.5 Synaptic Plasticity Dynamics for Deep Continuous Local Learning (Decolle)

Synaptic Plasticity Dynamics for Deep Continuous Local Learning [156] allows for online learning by employing local error functions. This property

makes this learning rule appropriate for neuromorphic hardware since it doesn't require any memory overhead. Synaptic plasticity rules are obtained systematically from user-defined cost functions and neuronal dynamics using machine learning frameworks' current auto differentiation algorithms. Back Propagation Through Time (BPTT) is not biologically realistic because neurons make their computations locally. Also, the continuous-time dynamics of SNNs raise a temporal credit assignment problem.

### 5.5.1 How it works

Decolle can compute gradients locally by using layerwise local readouts [157]. The temporal dynamics are handled by the established equivalence of SNNs and recurrent ANNs as mentioned in the Surrogate Gradient Learning section [150]. The plasticity rule is temporally localized because Decolle is written in such a way that the information needed to compute the gradient is carried forward. This idea is not entirely new, but similar methods require dedicated state variables for every synapse, thus scaling at least quadratically with the number of neurons, such as the SuperSpike with local learning rules . Decolle scales linearly with the number of neurons, which requires orders of magnitude less memory. Using readouts, a temporally and spatially local cost function. The authors point out the resemblance with readout mechanisms used in Liquid State Machines [158]. Readout neurons in liquid state machines can learn to extract information from certain microcircuits and report this information to other circuits.Readouts in Decolle are performed over a fixed random combination of neuron outputs. The authors also consider it a type of synthetic gradient with random initialization of the local readout.

### 5.5.2 Advantages of Decolle over previous methods

Decolle, like SuperSpike, uses surrogate gradients to update weights, but unlike SuperSpike, the cost function is local in time and space, requiring only one trace per input neuron. This allows the method to scale in space linearly. Furthermore, the computation of the gradients in Decolle can reuse the variables computed for the forward dynamics, resulting in no additional memory burden during learning. Decolle's local readout functions as a decoder layer and learns the internal weights with the readout weights being random and fixed. Internal weights training allows the network to learn representations that make it easier for succeeding layers to classify inputs [157] . This is

somewhat similar to reservoir networks but the readout weights are trainable, Decolle avoids this to reduce computation costs while still managing to achieve great classification accuracy. A Reservoir-based Convolutional Spiking Neural Network has been used to train a Dynamic Vision Dataset with great accuracy [159]. We point out again that SNNs are recurrent even when all the connections are feed-forward because the neurons maintain a state that is propagated forward at every time step. Because the full sequence and resulting activity states are stored to compute gradients, BPTT-based techniques can compute gradients gradients well, but at a cost in memory. Additionally, in SNNs we have to use a very large number of timesteps to capture the temporal dynamics of the input (a simulation time of 1ms or less is advised). This obviously means we have to use a large truncation window depending on the frequency of salient events in the input data(1000 timesteps if these occur every 1s).The size of SNN trainable by BPTT is severely limited by the available GPU memory [160] . Decolle requires an order T less memory resources compared to BPTT, where T is the sequence length.

### 5.5.3 Neuron and Synapse Model

The neuron and synapse models used in Decolle's paper experiments follow leaky, current-based integrate-and-fire dynamics with a relative refractory mechanism. We have defined this neuron model previously, but we define it again to allow us to explain easily the equations used for learning.The behavior the membrane potential $U_i$ of a neuron $i$ is described by the following differential equations:

$$\begin{aligned} U_i(t) &= V_i(t) - \rho R_i(t) + b_i \\ \tau_{mem}\frac{\mathrm{d}}{\mathrm{d}t}V_i(t) &= -V_i(t) + I_i(t) \\ \tau_{ref}\frac{\mathrm{d}}{\mathrm{d}t}R_i(t) &= -R_i(t) + S_i(t) \end{aligned} \tag{5.33}$$

$S_i(t)$ has a binary value (0 or 1) representing whether neuron $i$ spiked at time $t$ . The membrane potential is separated into two variables U and V is biologically inspired and represents a two compartment model, one dendritic (V) and one somatic (U) compartment .When the membrane potential reaches a threshold a spike is generated $S_i(t) = \Theta\left(U_i(t)\right)$, where $\Theta(x) = 0$ if $x < 0$, otherwise 1 is the unit step function. The constant $b_i$ represents the

intrinsic excitability of the neuron. The refractory mechanism is described by $R_i$ .Basically the neuron inhibits itself after firing, by a constant weight $\rho$.In other words, after a neuron has emitted a spike it won't excite as easily as before but can still emit a spike given a strong enough input. This behavior is governed by the last differential equation above. Usually in Integrate-and-Fire models the neuron cannot emit a spike immediately after even with a strong input. The factors $\tau_{ref}$ and $\tau_{mem}$ are time constants of the membrane and refractory dynamics.$I_i$ describes the total synaptic current of neuron $i$, expressed as:

$$\tau_{syn}\frac{\mathrm{d}}{\mathrm{d}t}I_i(t) = -I_i(t) + \sum_{j\in\mathrm{pre}} W_{ij}S_j(t) \tag{5.34}$$

$W_{ij}$ are the synaptic weights between pre-synaptic neuron $j$ and post-synaptic neuron $i$. $V_i$ and $I_i$ are linear with respect to the weights $W_{ij}$, so the authors of Decolle rewrite $V_i$:

$$\begin{aligned} V_i(t) &= \sum_{j\in\ \mathrm{pre}} W_{ij}P_{ij}(t) \\ \tau_{mem}\frac{\mathrm{d}}{\mathrm{d}t}P_{ij}(t) &= -P_{ij}(t) + Q_{ij}(t) \\ \tau_{syn}\frac{\mathrm{d}}{\mathrm{d}t}Q_{ij}(t) &= -Q_{ij}(t) + S_j(t) \end{aligned} \tag{5.35}$$

The state $P$ describes the traces of the membrane and $Q$ describes the traces of the current-based synapsey. For each incoming spike, the trace $Q$ undergoes a jump of height 1 and otherwise decays exponentially with a time constant $\tau_{\mathrm{syn}}$. The Post-Synaptic-Potentials (PSPs) of neuron $i$ caused by input neuron $j$ emerged by the weighting of the trace $Q_{ij}$ with the synaptic weight $W_{ij}$ . $P_{ij}$ and $Q_{ij}$ are only driven by $S_j$, and so the index $i$ is dropped. As a result, the $P$ and $Q$ variables are as many as are pre-synaptic neurons, independently of the number of synapses. However,for computer simulation we need the equations in discrete time, the simulation time step is denoted with $\Delta t$. The superscript $l$ represents the layer to which the neuron belongs.

The above equations are rewritten as such:

$$\begin{aligned} U_i^l[t] &= \sum_j W_{ij}^l p_j^l[t] - \rho R_i^l[t] + b_i^l \\ S_i^l[t] &= \Theta\left(U_i^l[t]\right) \\ P_j^l[t+\Delta t] &= \alpha P_j^l[t] + (1-\alpha) Q_j^l[t] \\ Q_j^l[t+\Delta t] &= \beta Q_j^l[t] + (1-\beta) S_j^{l-1}[t] \\ R_i^l[t+\Delta t] &= \gamma R_i^l[t] + (1-\gamma) S_i^l[t] \end{aligned} \tag{5.36}$$

The constants $\alpha = \exp\left(-\frac{\Delta t}{\tau_{\text{mem}}}\right), \gamma = \exp\left(-\frac{\Delta t}{\tau_{\text{ref}}}\right)$, and $\beta = \exp\left(-\frac{\Delta t}{\tau_{syn}}\right)$ reflect the decay dynamics of the membrane potential $U$, the refractory state $R$ and the synaptic state $Q$ during a $\Delta t$ timestep.

### 5.5.4 Deep Learning

A global cost function is assumed: $\mathcal{L}\left(S^N\right)$ . It is defined on the spikes $S^N$ of the top layer and targets $\hat{Y}$, the gradients with respect to the weights in layer $l$ are:

$$\frac{\partial \mathcal{L}\left(S^N\right)}{\partial W_{ij}^l} = \frac{\partial \mathcal{L}\left(S^N\right)}{\partial S_i^l} \frac{\partial S_i^l}{\partial U_i^l} \frac{\partial U_i^l}{\partial W_{ij}^l} \tag{5.37}$$

The factor $\frac{\partial \mathcal{L}\left(S^N\right)}{\partial S_i^l}$ describes the backpropagated errors. The backpropagated errors represent how the output of neuron $i$ in layer $l$ modifies the global loss. This problem is known as the credit assignment problem. However, it usually involves non-local terms,the activity neurons, their errors, and how they behaved in the past. Taking into consideration the non-locality a hidden neuron cannot predict how modifying its own behavior will affect the top-layer cost. Approximations to the backpropagated errors in SNNs can allow local learning,ex. in feedback alignment . However, maintaining the history of the dynamics efficiently remains a challenging and open problem. What remains an open problem is how to store the past dynamics efficiently and this one of the main achievements of Decolle, reduced GPU memory requirement compared to using BPTT methods.

Decolle focuses on a type of deep local learning in which random readouts are attached to deep layers and auxiliary cost functions are defined over the readout 5.13.

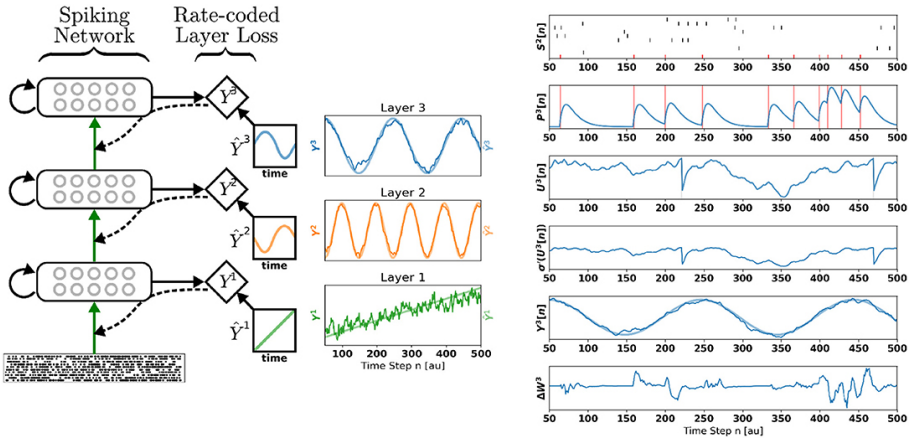



Figure 5.10: Readout Mechanism .

For neurons in the deep layers, these auxiliary cost functions provide a task-relevant source of error. By multiplying the neural activations with a random and fixed matrix, the random readout is obtained. Deep layer training with auxiliary local errors that reduce cost locally allows the network as a whole to achieve a low top layer cost. As discussed in [161], reducing the classification loss of a local readout puts pressure on deep layers to acquire important task-relevant features, allowing the random local classifiers to accurate enough for the global loss to get reduced. Furthermore, each layer draws on the features of the preceding layer to learn features for its local random classifier that are more disentangled with regard to the categories than the prior layer. In this article, we focus on the fact that, provided local loss functions, surrogate learning in deep spiking neural networks becomes particularly efficient. This results in an efficient surrogate learning approach for deep spiking neural networks.

### 5.5.5 Decolle Learning Rule

The random readout that is attached to each of the $N$ layers of spiking neurons is given by:

$$Y_i^l = \sum_j G_{ij}^l S_j^l \tag{5.38}$$

where $G_{ij}^l$ are fixed, random matrices (one for each layer $l$ ) and $\Theta$ is an activation function. The global loss function is then defined as the sum

of the layerwise loss functions defined on the random readouts, i.e. $\mathcal{L} = \sum_{l=1}^{N} L^l\left(Y^l\right)$. To enforce locality, Decolle sets to zero all non-local gradients, i.e., $\frac{\partial L^l}{\partial W_{ij}^m} = 0$ if $m \neq l$. The weight updates at each layer are:

$$\Delta W_{ij}^l = -\eta \frac{\partial L^l}{\partial W_{ij}^l} = -\eta \frac{\partial L^l}{\partial S_i^l} \frac{\partial S_i^l}{\partial W_{ij}^l} \tag{5.39}$$

where $\eta$ is the learning rate. Assuming the loss function depends only on variables in same time step, the term $\frac{\partial L^l}{\partial S_i^l}$, can be computed using the chain rule of derivatives. Applying the chain of derivatives to the second gradient term yields:

$$\frac{\partial S_i^l}{\partial W_{ij}^l} = \frac{\partial \Theta\left(U_i^l\right)}{\partial U_i^l} \frac{\partial U_i^l}{\partial W_{ij}^l} \tag{5.40}$$

Due to the sparse, binary activation of spiking neurons, this expression vanishes everywhere except at 0 , where it is infinite . To solve this problem Decolle uses surrogate gradient-based learning:

$$\frac{\partial S_i^l}{\partial W_{ij}^l} = \sigma'\left(U_i^l\right) \frac{\partial U_i^l}{\partial W_{ij}^l} \tag{5.41}$$

where $\sigma'\left(U_i^l\right)$ is the surrogate gradient of the non-differentiable step function $\Theta\left(U_i^l\right)$.

$$\frac{\partial U_i^l}{\partial W_{ij}^l} = P_j^l - \rho \frac{\partial R_i^l}{\partial W_{ij}^l} \tag{5.42}$$

The terms involving $R_i^l$ are difficult to calculate because they depend on the spiking history of the neuron. As in Superspike, terms involving $R_i^l$ are ignored. Regularization is applied to favor low firing rates, this causes the term $R_i^l$ to have negligible effect. The Decolle rule that describes how the synaptic weights are updated is:

$$\Delta W_{ij}^l = -\eta \frac{\partial L^l}{\partial S_i^l} \sigma'\left(U_i^l\right) P_j^l \tag{5.43}$$

In the case of the Mean Square Error (MSE) loss for layer $l$, described as:

$$L^l = \frac{1}{2} \sum_i \left(Y_i^l - \hat{Y}_i^l\right)^2 \tag{5.44}$$

the Decolle rule becomes:

$$\begin{aligned} \Delta W_{ij}^{l} &= -\eta \, \text{error}_i^l \, \sigma' \left( U_i^l \right) P_j^l \\ \text{error}_i^l &= \sum_k G_{ki}^l \left( Y_k^l - \hat{Y}_k^l \right) \end{aligned} \tag{5.45}$$

where $\hat{Y}^l$ is the pseudo-target vector for layer $l$.

$$\mathcal{L}_g = \sum_l L^l + \lambda_1 \left\langle \left[ U_i^l + 0.01 \right]^+ \right\rangle_i + \lambda_2 \left[ 0.1 - \left\langle U_i^l \right\rangle_i \right]^+ \tag{5.46}$$

### 5.5.6 Computational Complexity

The variables P and U required for learning are local and available from the forward dynamics. Because the errors are computed locally to each layer, DECOLLE does not need to store any additional intermediate variables. The computational cost of the weight update is one addition and two products per connection. This makes DECOLLE significantly cheaper to implement compared to BPTT for training SNNs which scales spatially as O(NT),T is the number of timesteps. Decolle employs auto-differentiation tools; all that is required is backpropagation across a subgraph corresponding to one layer in the same time step. Because the information needed to compute the gradients is carried forward in time and local loss functions provide the gradients needed for each layer. Exact implementation details of auto differentiation is given in Decolle's paper footnotes.

### 5.5.7 Our reflection

When compared to BPTT, Decolle is a very useful learning approach because it has orders of magnitude less space complexity. Decolle is merely a temporary solution for training datasets in neuromorphic hardware until we figure out the more clever and complicated modulatory signals our brains use.There is also the issue of losing important information/features in longer sequences of input data which e-prop seems better able to handle such data. Furthermore, when compared to a mammalian brain, present neuromorphic hardware is quite primitive. Given our poor understanding of how modulatory signals exploit the brain's intricate 3D structure and vast number of neurons and synapses, existing learning approaches for spiking neural networks are also very primitive.Learning methods will have to follow our

understanding of the mammalian brain, which is affected by available brain imaging technology, unless someone can beat billions of years of evolution.

## 5.6 Eligibility Propagation (e-Prop) - A solution to the learning dilemma for recurrent networks of spiking neurons

Feed-forward networks, while showing remarkable results, do not seem to follow the biological model since the brain is shown to consist of numerous recurrently connected cells. Artificial recurrent networks have demonstrated noteworthy results in temporal data, while recurrent spiking neural networks underlie the astounding information processing capabilities of the brain. However, the lack of understanding of the process of learning in biological brains leads to the use of non-biologically plausible algorithms. To counteract this fact, supported by neuroscientific findings, Eligibility Propagation (e-Prop) [162], an algorithm for training recurrent spiking neural networks (RSNNs) was developed. It gives the network the ability of online training while approaching the performance of Back Propagation Through Time (BPTT) algorithm, the best-known method for RNN training.

### Learning Scheme

In order to train a Neural Network, most learning methods focus on minimizing a loss function E. The method of backpropagation is widely used in feed-forward neural networks, where gradients of this loss function propagate backwards through the network calculating each weight's (neuron's synapse) contribution to the loss function. Recurrent models can still use this method, by unfolding themselves through time and in each step a copy of the network is created and the loss function propagates backwards through time (Back Propagation Through Time - BPTT). This method can and has been used in spiking recurrent networks in the past successfully. But it is easily understood that it requires storing the intermediate states of the neurons in each timestep, while the learning process itself happens offline. These two factors suggest that the brain probably does not use BPTT for learning tasks.

It has been found, however, that neurons seem to remember faintly certain events (event order), e.g. in the form of calcium ions, that are known

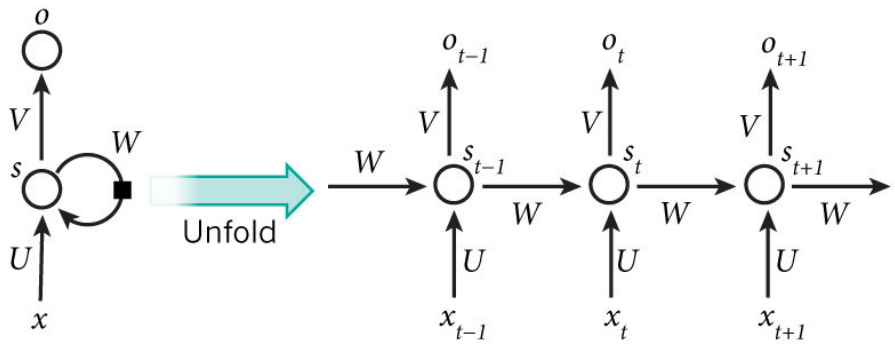



Figure 5.11: The unfolding of a recurrent neural network to be trained with backpropagation. Each state s is stored in each timestep with its corresponding input and output.

to induce synaptic plasticity - these traces are often called eligibility traces. Neurotransmitters have been shown to act as learning signals in populations of neurons regarding behavioral results. This has led to the belief that neurons train without the need for backpropagation but they use the eligibility traces of the neuron to converge to local optimal. The e-prop algorithm is a novel algorithm based on this idea.

The problem to be tackled is, as in most neural network, to minimize a loss function $E$ by teasing the recurrent network's weights $W$. The nature of the loss function can differ depending on the task, for example it can measure the deviation of the network's output from an optimal output (for task like classification or regression). The value of the gradient of the loss function over a given synapse (weight) $\frac{dE}{dW_{ji}}$ gives the proportional change in value of the synapse for the steepest decrease of the loss function regarding the change of this synapse. In spiking RNNs, the spike of a neuron j at a given time t is denoted as binary variable $z_j^t$. As a result $z_j^t$ is non-differentiable. Huh Dongsung and Sejnowski Terrence J. in their work [163] proposed a pseudo derivative for spikes so as to avoid the non-differentiability problem of the nature of the spike. This manipulation gives the ability to differentiate the spike of the neuron j $z_j^t$ over the synapses of the neuron. This local derivative collects information about the gradient of the synapse over the loss function for time t, analogous to the eligibility trace of biological neurons. This way, eligibility trace $e_j^t i$ is defined in eq. 4.63.

$$e_{ji}^{t} \stackrel{\text{def}}{=} \left[\frac{dz_j^t}{dW_{ji}}\right]_{\text{local}} \tag{5.47}$$

Eligibility trace is related to the hidden layer $\mathbf{h}_j^t$ in time t. While it holds the most possible amount of information computed locally about the relative participation in the loss function, its value is dependant and can be updated using the eligibility trace of the synapse in the last time step and the state of the hidden layer of neurons. This key factor enables the ability of online learning since it does not require backpropagation in distant previous states in order to train the network (alter the synapse). As it is understood, this model seems to mimic the biological neuron learning than previous proposed methods such as BPTT, but it still does not factor in slow-changing variables of the neurons that affect the procedure of plasticity and as a result the learning. However, in the same paper [162] a mathematical model is presented that takes into account such behaviors and will be presented later in this chapter. This manipulation of the spike $z_j^t$ and its differentiation gives the ability to calculate the derivative of the loss function over a synapse as a sum of all the timesteps of the product of the derivative of the loss function over each spike and the derivative of the spike of a given neuron over its synapses, i.e. the eligibility trace, according to the chain rule, as presented in equation 4.64.

The second product is, as stated before, is the eligibility trace and replacing the eq. 4. 63 we arrive at the representation of the gradient of the loss function due to the synapse as shown below:

$$\frac{dE}{dW_{ji}} = \sum_t \frac{dE}{dz_j^t} \cdot \left[\frac{dz_j^t}{dW_{ji}}\right]_{\text{local}} \quad \Longleftrightarrow \quad \frac{dE}{dW_{ji}} = \sum_t \frac{dE}{dz_j^t} \cdot e_{ji}^t \tag{5.48}$$

In equation 4.65 it can be seen that the term of the eligibility trace does not depend on the loss function $E$ while only the first product of the sum does. $\frac{dE}{dz_j^t}$ is defined as the learning signal (eq. 4.65) and in the biological neuron represents the learning signals like the ones that were aforementioned e.g. neurotransmitters, adding to the biological plausibility of the model. As a result eq. 4.64 can now be written replacing the first product term with the eq. 4.65 resulting in equation 4.66 that follows.

$$L_j^t \stackrel{\text{def}}{=} \frac{dE}{dz_j'} \tag{5.49}$$

$$\frac{dE}{dW_{ji}} = \sum_t L_j^t e_{ji}^t \tag{5.50}$$

The learning signal term, defined above, plays the sole role in connecting the loss function with the spike emission of a neuron.

Summing up the e-prop learning scheme, in contrast to unfolding the recurrent spiking neural network and computing gradients of the loss function for every copy, as it is done in BPTT, e-prop uses eligibility traces of synapses that hold information about previous activations of neurons and using learning signals, that are correlated with the spike emission and the loss function itself, compute in an online way the desired gradient. In figure 4.23 a graphical explanation of the difference between the BPTT and e-prop learning algorithms is given. While BPTT requires all previous states of the network to be stored, calculate and pass information in order for the network to learn (using an offline process), e-prop only requires the previous eligibility traces of each synapse (weight) and the learning signal $L_j^t$ to calculate the ideal weight change of each synapse (online process) to minimize the distance over its target (i.e. minimize the loss function).

In a feed-forward network, where the hidden and output states are synchronized, the learning signal captures the full influence of the spike $z_j^t$ to the loss function. But in recurrent networks, the emission of the spike might not influence the output and as a result the loss function in the same timestep but later in time. This leads to the learning signal as presented above to not capture the precise gradient of the loss function due to the spike emitted. To deal with the resulting problem, the authors propose the use of an approximation of the learning signal (using the operator of partial derivative - $\frac{\partial E}{\partial z_j^t}$ to distinguish the two values). The justification of the use of this incomplete learning signal is that, while it does not account for the future influence of the spike, the combination with the eligibility trace gives rise to the ability of the neuron to reach back into the past, making the neuron capable of learning temporal assignments. In figure 4.24 are shown the computational dependencies between the network's elements. It is shown that the eligibility trace does depend on past and present events while the learning signal requires the knowledge of future events. It is also shown the dependence of the aforementioned learning signal approximation, which is only dependent on the loss function at the calculation time. This way it is clear that the network is capable of online training.

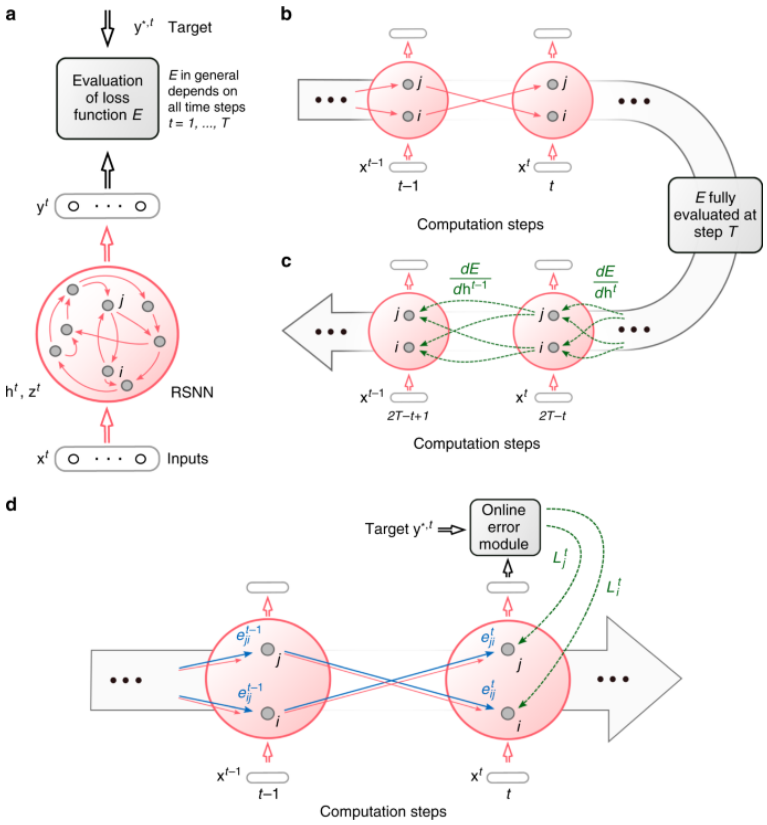



Figure 5.12: The difference between BPTT and e-prop training. **A.** the architecture of the RSNN. $x, h, z, y$ represent the input, hidden state, spike and output of the network while the exponent $t$ represents the timestep of each variable. **B.** Unfolded version of the network shown for two timesteps, t and t-1. The recurrent synapses are now unfolded and connect the two new networks through time. **C.** BPTT requires the loss gradients to be backpropagated backwards in time. **D.** E-prop online learning requires only previous eligibility traces of synapses and learning signals that are calculated using the loss function in the desired timestep.

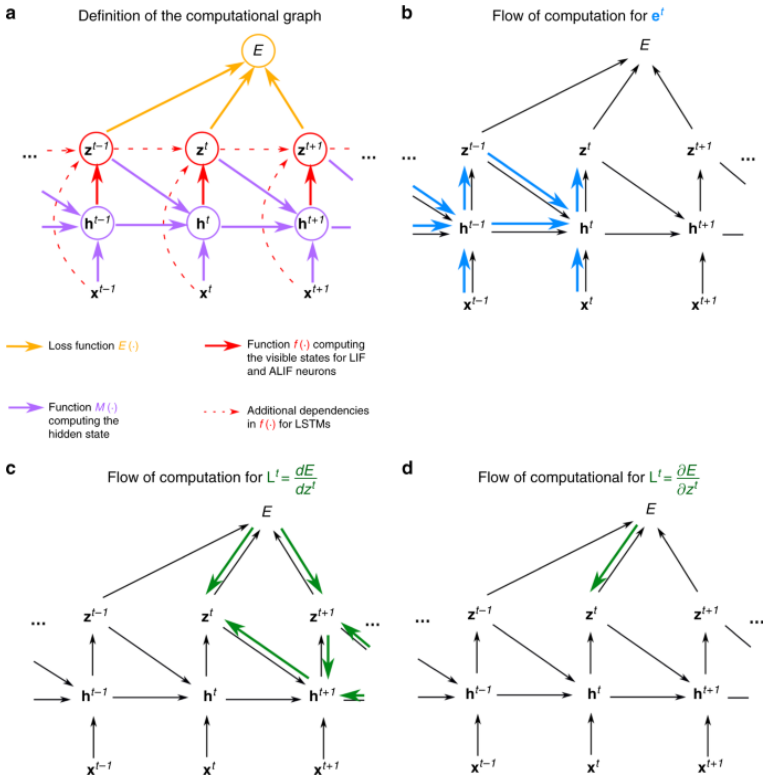



Figure 5.13: Computational flow of signals of e-prop. **A.** The network unfolded through time and the dependencies between input $\mathbf{x}^t$, hidden state $\mathbf{h}^t$, neurons' output $\mathbf{z}^t$ and loss function $\mathbf{E}$ drawn in arrows. **B.** Computational flow of eligibility traces. **C.** Computational flow of the ideal learning signal $L^t$. The dependencies exist in present and future instances. **D.** The computational flow of the approximation of the learning signal $\frac{\partial E}{\partial z_j^t}$. As it is shown, it is only dependent to present events and is computed in timestep t.

In e-prop learning rule, the learning signal $L_j^t$ is calculated as the distance or deviation of the output of the network $y_k^t$ to the desired output $y_k^{*,t}$. More specifically, each output neuron of the network propagates the deviation with the desired output to the neuron j through a weighted synapse $B_{jk}$. Therefore, the value of the learning signal is calculated in the following equation:

$$L_j^t = \sum_k B_{jk} \underbrace{\left(y_k^t - y_k^{*,t}\right)}_{\text{deviation of output } k} \tag{5.51}$$

The choice of the weight $B_{jk}$ is left to the engineer of the network while the authors propose three possible choices. Symmetric e-prop where the weight $B_{jk}$ is equal to the weight of the synapse j with the output k, $W_{kj}^{out}$, Random e-prop where the weights $B_{jk}$ are randomly chosen and remain fixed (this implies connections through $B_{jk}$ for neurons of the hidden layer with the output layer, where, in the network, are not connected), and Adaptive e-prop which is like Random e-prop but the weights $B_{jk}$ evolve through simple plasticity rules. The last two proposals seem to be biologically plausible compared to the first one, according to the authors.

### 5.6.1 Mathematical basis

Consider a recurrently connected spiking neural network with input $\mathbf{x}^{\mathrm{t}}$, hidden state $\mathbf{h}^{\mathrm{t}}$ and neurons' output $\mathbf{z}^{\mathrm{t}}$. In recurrent networks the hidden state at time t is dependent on the previous hidden state, the input at time t and the neurons' output in the previous timestep. So it can be represented as the following function M: $\mathbf{h}_j^t = M\left(\mathbf{h}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \mathbf{W}_j\right)$. The function $f$ is defined as $z_j^t = f\left(\mathbf{h}_j^t\right)$ which describes the update of the state of a neuron $j$. As stated before, in recurrent network events that happen in timestep t may affect the loss function through their influence of future states of the neurons' hidden states. This causes the need of defining two types of derivatives, denoted as $\frac{dA}{dB^t}$ and $\frac{\partial A}{\partial B^t}$, where the first one denotes the dependence of $A$ to $B^t$ directly and indirectly through the influence of $B^t$ to other variables $B^{t+\tau}$ in future timesteps and the second one the dependence of $A$ to $B^t$ directly in time t.

In a recurrent neural network, calculating the gradient of $E$ over $W_{ji}$ over a time t is described by the equation.

$$\frac{dE}{dW_{ji}} = \frac{dE}{dh_j^t}\frac{\partial h_j^t}{\partial W_{ji}^t} \tag{5.52}$$

Unfolding the network, as in BPTT, the summed up derivatives through time yield the following equation, which describes the BPTT of the losses. The eq. 4.69 is commonly used in BPTT methods where each layer $l$ of the unfolded network is equivalent with the instance of time t of the RSNN presented here.

$$\frac{dE}{dW_{ji}} = \sum_{t'} \frac{dE}{d\mathbf{h}_j^{t'}} \cdot \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \tag{5.53}$$

From fig. 4.24, applying the chain rule of the computational graph, the derivative $\frac{dE}{dh_j^t}$ can be expressed as the derivative of the next timestep as shown in equations 4.70 and 4.71. In equation 4.70 the derivative $\frac{dE}{dz_j^t}$ is replaced with the learning signal $L_j^t$ from eq. 4.65.

$$\frac{dE}{d\mathbf{h}_j^t} = \frac{dE}{dz_j^t}\frac{\partial z_j^t}{\partial \mathbf{h}_j^t} + \frac{dE}{d\mathbf{h}_j^{t+1}}\frac{\partial \mathbf{h}_j^{t+1}}{\partial \mathbf{h}_j^t} \tag{5.54}$$

$$= L_j^t \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} + \frac{dE}{d\mathbf{h}_j^{t+1}}\frac{\partial \mathbf{h}_j^{t+1}}{\partial \mathbf{h}_j^t} \tag{5.55}$$

Replacing the first product term of eq. 4.69 with 4.71 we get the eq. 4.72. Applying the same method and replacing the new derivatives of the loss function over the hidden states in future timesteps, we get the equation 4.73

$$\frac{dE}{dW_{ji}} = \sum_t \left( L_j^t \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} + \frac{dE}{d\mathbf{h}_j^{t+1}}\frac{\partial \mathbf{h}_j^{t+1}}{\partial \mathbf{h}_j^t} \right) \cdot \frac{\partial \mathbf{h}_j^t}{\partial W_{ji}} \tag{5.56}$$

$$= \sum_t \left( L_j^t \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} + \left( L_j^{t+1} \frac{\partial z_j^{t+1}}{\partial \mathbf{h}_j^{t+1}} + (\cdots) \frac{\partial \mathbf{h}_j^{t+2}}{\partial \mathbf{h}_j^{t+1}} \right) \frac{\partial \mathbf{h}_j^{t+1}}{\partial \mathbf{h}_j^t} \right) \cdot \frac{\partial \mathbf{h}_j^t}{\partial W_{ji}} \tag{5.57}$$

In the above expression, expanding the parenthesis results in a double sum where each learning signal $L_j^t$ is multiplied by some factor which, as it

will be shown, is equal to the eligibility trace. It should be noticed that the term $\frac{\partial \mathbf{h}_j^{t+1}}{\partial \mathbf{h}_j^t}$ represents events on the neuron $j$ that occur until time t without the effect of any future event or losses. Collecting them all together, as shown in eq. 4.75 results in a product that is defined by past events and gives rise to the idea of e-prop and the eligibility trace.

$$\frac{dE}{dW_{ji}} = \sum_{t'} \sum_{t \geq t'} L_j^t \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \cdots \frac{\partial \mathbf{h}_j^{t'+1}}{\partial \mathbf{h}_j^{t'}} \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \tag{5.58}$$

$$= \sum_{t} L_j^t \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \sum_{t \geq t'} \frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \cdots \frac{\partial \mathbf{h}_j^{t'+1}}{\partial \mathbf{h}_j^{t'}} \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \tag{5.59}$$

The inner sum of the right-hand part of the equation is defined as the eligibility vector $\epsilon_{\mathbf{ji}}^{\mathbf{t}}$ which satisfies the recursive expression 4.77.

$$\epsilon_{\mathbf{ji}}^{\mathbf{t}} \stackrel{\text{def}}{=} \sum_{t \geq t'} \frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \cdots \frac{\partial \mathbf{h}_j^{t'+1}}{\partial \mathbf{h}_j^{t'}} \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \tag{5.60}$$

$$\epsilon_{ji}^t = \frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \cdot \epsilon_{ji}^{t-1} + \frac{\partial \mathbf{h}_j^t}{\partial W_{ji}} \tag{5.61}$$

By definition of the eligibility trace from eq. 4.63, applying the chain rule results in equation 4.78, where, by replacing the sum with the eligibility vector, the calculation of computing the eligibility trace is given. Equation 4.77 provides the online learning of the e-prop method in calculating the eligibility trace through equation 4.79

$$e_{ji}^t \stackrel{\text{def}}{=} \left[ \frac{dz_j^t}{dW_{ji}} \right]_{\text{local}} = \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \sum_{t \geq t'} \frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \cdots \frac{\partial \mathbf{h}_j^{t'+1}}{\partial \mathbf{h}_j^{t'}} \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \tag{5.62}$$

$$e_{ji}^t = \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \cdot \epsilon_{ji}^t \tag{5.63}$$

Thus, replacing the eligibility trace from eq. 4.78 to 4.75 the proof of equation 4.66 is completed.

### 5.6.2 Reward-based e-Prop

An important method of learning in neural networks is reinforcement learning (RL). Deep RL cleverly exploits the power of BPTT to achieve impressive results. As a result, the e-prop method can be applied for RL. The authors of [162] propose a biologically plausible method of reinforcement learning using the e-prop scheme, which is based on the combination of policy gradient and the actor-critic method. One noteworthy difference of this method (reward-based e-prop) compared to the e-prop discussed before is that in the previous case the learning signal depended on the deviation of an external signal, while in this case, the learning signal carries information about how the chosen action diverges from the action proposed by the network.

In order to calculate the appropriate weight change, a fading memory - low pass filter notation has to be defined. The operator $\mathcal{F}_\alpha$ denotes a low pass filter that satisfies the following equation.

$$\mathcal{F}_\alpha\left(x^t\right) = \alpha\mathcal{F}_\alpha\left(x^{t-1}\right) + x^t \tag{5.64}$$

with the initial condition $\mathcal{F}_\alpha\left(x^0\right) = x^0$. A simplified notation for the low-pass filter application to the eligibility trace $e_{ji}^t$ is given as $\mathcal{F}_\kappa\left(e_j\right)^t = \bar{e}_{ii}^t$, where $\kappa$ represents the leakage of the neuron between consecutive timesteps. The proposed plasticity rule based on e-prop, which integrates online learning is based on the appliance of a low-pass filter $\mathcal{F}_\gamma$, representing fading memory, in the product $L_j^t\bar{e}_{ji}^t$. The result is multiplied by the prediction error $\delta^t = r^t + \gamma V^{t+1} - V^t$, with $r^t$ being the reward at time t. The calculation of the weight change at time t is given by the calculated result multiplied by a forgetting factor $\eta$ and so, the expression is presented in eq. 4.81

$$\Delta W_{ji}^t = -\eta\delta^t\mathcal{F}_\gamma\left(L_j^t e_{ji}^t\right) \tag{5.65}$$

This novel approach of the weight change offers better results than previous attempts with the use of eligibility trace. The addition of the learning signal and the low-pass filter in the computation results in less noisy gradient estimates and, as the authors have proven, show similar behavior with BPTT used in deep reinforcement learning.

The authors tested this reward-based e-prop method on two Atari games, Pong, a frequently used game as a basis for RL, and Fishing Derby, a game with a high need of temporal processing. They managed to train a network in an online manner without the need for experience replay with perfect

memory. They strictly used online training with each frame as input and increasing episode lengths with adaptive learning rate. This way, in early short episodes, the network could learn useful skills while, in longer episodes, fine-tune those skills. The networks showed competitive results with the Fishing Derby LSNN performance being similar to reference offline algorithms applied to LSTM networks.

### 5.6.3 Further reading

The idea of e-prop algorithms has attracted the interest of the research community. Zixuan Zhao et. al. in their work [164] present an implemented library named Neko for training Spiking Neural Networks in python, where the e-prop method has been implemented. In addition, they have also coded the adaptive LIF neuron (ALIF) for the network as presented in the work of Guillaume Bellec et. al. [162], giving the ability for researchers to easily study this method.

As stated before, e-prop provides a biologically plausible model of training neural networks. However, Manuel Traub et. al. in their work [165] showed that the proposed model presented above does not take into account STDP, a process that has been described in 4.3.2 and is used by the brain for neural plasticity (chapter 2.3). They prove that the neuron models that were used (LIF and A-LIF neuron models) do not reflect STDP behavior in gradients based on eligibility trace. They show that more complex models such as the Izhikevich neuron model carry the needed information for the STDP behavior to be present in e-prop. Furthermore, they propose an altered LIF neuron model named STDP-LIF neuron that acts in the desired way for STDP to be present through the learning process. The discrete differential equation for the STDP-LIF neuron is

$$u_j^{t+1} = \alpha u_j^t + I_j^t - z_j^t \alpha u_j^t - z_j^{(t-\delta t_{ref})} \alpha u_j^t \tag{5.66}$$

where the $u_j^t$ is the membrane voltage of the neuron j at time t, $z_j^t$ represents the presence of the spike of the neuron j at time t and $\delta t_{ref}$ is the refractory period of the neuron. This way, the effect of the spike of the neuron is present in the hidden state as it is a part of the equation of the derivative in the hidden state. This presence is responsible for the STDP behavior of the network, as the authors prove in their work. Applying the STDP rule gives rise to the ability of the network for precise spike timing of the net-

work, as the authors have shown where STDP-LIF neurons performed better compared to LIF neurons in the aforementioned task, trained with e-prop, having as input a Poisson distributed spiketrain of frequency 25 Hz.

In continuation of the study of the e-prop training algorithm, this study has been published [166] where the neuron models presented as far are compared. The author implements the STDP-A-LIF neuron model which, as it is shown performs better than the A-LIF model. However, the Izhikevich neuron model was unstable and performed worse than the alternatives, showing that the existence of the STDP behavior in e-prop does not guarantee better results by itself, although its addition in non-STDP like models seems to improve results. Furthermore, this study shows that multilayer recurrent spiking networks increase the instability of the convergence of the network and perform worse than single-layer networks.

Another important feature that neural networks aim for is one-shot learning. Compared to typical learning methods, where the learning process takes hundreds or thousands of repetitions of the task, o one-shot learning aims to learn information about the task in a single or only a few repetitions. As this paper shows [167], e-prop cannot achieve this effect. Nevertheless, the authors propose a method for spiking neural networks using the e-prop method to achieve one-shot learning, adding to the biological plausibility of the network, mimicking the operation of the brain. In the proposed method, a second optimized RSNN is used to emit appropriate learning signals. This way, the problem of the missing future information in the learning signal is tackled and in a way resolved. This method, named natural e-prop, is tested and the produced results show improvement over the previous e-prop versions.

# Chapter 6

# Dynamic Vision Sensors

Dynamic Vision Sensors or Event cameras are bio-inspired sensors that operate in a somewhat different way than conventional cameras. They calculate per-pixel brightness changes asynchronously rather than collecting images at a fixed time. Dynamic vision sensors provide great advantages when compared to standard cameras. These include higher temporal resolution , very high dynamic range (140 dB compared to 60 dB). Also, they consume much less power, and they provide high pixel bandwidth which results in lower motion blur. These advantages over standard cameras can help robotics which require low-latency, high-speed, efficient cameras to function well and efficiently. DVS have also the potential to reduce training cost of Computer Vision deep neural networks while offering equal or better performance.However, because event cameras differ from ordinary cameras in that they measure per-pixel brightness variations (called "events") asynchronously rather than recording "absolute" brightness at a fixed rate, new methods are needed to analyze their output .

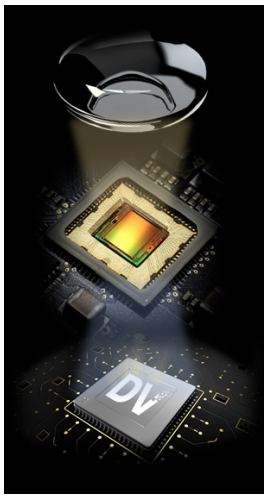



Figure 6.1: iniLabs Dynamic Vision Sensor Architecture

Some applications of these sensors are:

- Object Tracking
- Surveillance and monitoring

- Object/Gesture recognition
- depth estimation
- optical flow estimation
- Simultaneous Localization and Mapping field of research
- Image Deblurring

## 6.1 Event Cameras operating principles

How exactly do Event Cameras calculate per-pixel brightness changes? Each output event or spike represents a change of brightness(log intensity) depended on a threshold, at a particular time. Each pixel has to remember its own log intensity every time it "spikes" and is ready to send out an event again if a certain change in brightness is perceived. This is different to traditional cameras where information about the environment is recorded in frames per second(in cycles) instead of continuously monitoring the environment. An output event contains the $x,y$ location, time $t$ and the 1-bit polarity $p$ ( brightness increase "ON" or decrease "OFF, see also 6.2b,6.2e,6.2f [168]).

The events are exported from the camera using a shared digital output bus, usually by using address-event readout(AER) [169] [170] . The amount of data-points generated by those sensors depends on the amount of information that changes in time, in the environment. This is achieved by adapting the sampling rate to the rate of change of the log intensity signal. The time resolution is in the order of microseconds while latency is a little slower in the order of sub-milliseconds. This allows sensors to react quickly to visual stimuli and makes the sensors appropriate for autonomous driving. Addi-

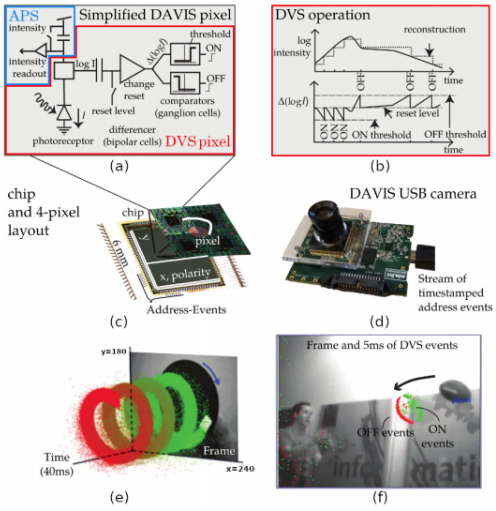



Figure 6.2: DAVIS camera

tionally, the DVS brightness change
events have a built-in invariance to scene illumination.

### 6.1.1 Event Sensors Devices

Apart from the original DVS event camera [171] there have been recent advancements [172] , [170], [173], [174] . One of the most widely used sensors is the Dynamic and Active Pixel Vision Sensor [168] 6.1.DAVIS combines a conventional active pixel sensor (APS) [175] in the same pixel with DVS.

### 6.1.2 Challenges

- Dealing with various space-time output: Event cameras' output differs significantly from conventional cameras: events are asynchronous and spatially sparse, whereas pictures are synchronous and dense. As a result, frame-based vision algorithms do not apply.
- Each event contains information about how brightness changes. However, brightness changes are not only related to current scene brightness but on current and past relative motion between the scene and the camera.
- Noise: Because of the intrinsic shot noise in photons and transistor circuit noise, all vision sensors are noisy.

### 6.1.3 Event Generation

An event sensor [171] has independent pixels that respond to changes in their log photocurrent *L=log(I)* . An event $e_{\mathrm{k}} = (x_{\mathrm{k}}, t_{\mathrm{k}}, p_{\mathrm{k}}$ is triggered at pixel $x_{\mathrm{k}} = (x_{\mathrm{k}}, y_{\mathrm{k}})^T$ at time $t_{\mathrm{k}}$ when the brightness increases a certain temporal contrast threshold $C$ since the last event.

$$\Delta L(x_{\mathrm{k}}, t_{\mathrm{k}}) = L(x_{\mathrm{k}}, t_{\mathrm{k}}) - (x_{\mathrm{k}}, t_{\mathrm{k}} - \Delta t_{\mathrm{k}}) \tag{6.1}$$

$$\Delta L(x_{\mathrm{k}}, t_{\mathrm{k}}) = p_{\mathrm{k}} C \tag{6.2}$$

Where $C > 0, \Delta t_{\mathrm{k}}$ is the time elapse since the last event at the same pixel, and the polarity $p_{\mathrm{k}} \in +1, -1$ is the sign of the brightness change [171]. The contrast sensitivity $C$ is determined by the pixel bias currents [176] [177],

which also set the speed and threshold of the change detector in 6.2. Typical DVS's thresholds are set 10-50 percent of illumination change. More advanced methods of generating events exist but is not the subject of this dissertation. Some of these methods include setting a threshold on magnitude of the brightness change since the last event happened and are used as a basis of physically grounded event based algorithms [178]. Events can also be set to be generated by moving edges. More complex models include taking into account sensor noise and transistor mismatch making these models probabilistic.

## 6.2 Event Processing

Event processing aims to extract useful information from the data, but is also depended on the type of task we are interested in. Approaches for event processing that work on an event-by-event basis, with the state of the system (estimated unknowns) changing with each arrival of a single event, resulting in the shortest possible latency do not provide enough information for estimation and so methods that operate on events in groups or packets are more appropriate.These however, result in some increased latency but still provide a state update on event arrival. Given that events are processed in an optimized framework, another distinction is the type of objective or loss function used: geometric vs. temporal vs. photometric (e.g., a function of event polarity or activity).

### 6.2.1 Event Representation Methods

These methods can be considered as the pre-processing step before the data is inserted into the desired neural network. The choice however is also influenced by the type of neural network that is used.

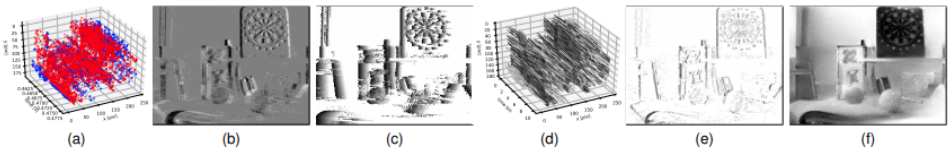



Figure 6.3: Event representations

(a) Events in space time(positive polarity is in blue color while negative polarity is in red).

(b) Events are accumulated in a 2-D image .This allows conventional computer vision algorithms to be applied. However, this method defeats our purpose of recording events in the time dimension, we are losing valuable information

(c) Time surface is a 2D map where each pixel stores a single time value. This is called Motion History Images in computer vision [179] .One example [180] events can be converted into an image that represents the recent motion history exposing the rich temporal information of the recorded data. Their effectiveness is degraded on textured scenes.

(d) Similar concept of (c) but now the 3rd dimension allows to store the whole sequence of events(only negative events are shown in the figure).

(e) Motion-Compensated event image [177] .A representation that is based on both events and motion hypotheses. The theory behind motion compensation is that as an edge moves over the image plane, it generates events on the pixels it passes through; the edge's motion may be calculated by warping the events to a reference time and optimizing their alignment, resulting in a sharp image (i.e., histogram) of warped events. This is mostly useful if someone wants to tracks features in a video.

(f) Reconstructed image intensity [181].

# Chapter 7

# Experiments

In the following chapter, we use the developed theory on Spiking Neural Networks to compare the performance of such networks in DVS datasets. The chosen task is in the category of Vision time-dependent task, where the networks have to recognize and classify gestures recorded with a Dynamic Vision Sensor (DVS - chapter 5). The choice of DVS data was preferred since Dynamic Vision Sensors present multiple advantages over RGB sensors, mainly in energy consumption, and have high potential in future uses.

## 7.1 Dataset

In order to take advantage of temporal information, we chose a dataset that has a temporal component on its own. In contrast with non-temporal datasets, like still images, that require either encoding schemes like the ones describes in chapter 4.2 to convert the data into spiketrains, or by constructing a DVS dataset of the images by moving them in front of a Dynamic Vision Sensor (e.g. DVS MNIST dataset [182]), a time-dependent dataset, like video, could fully exploit the features that Spiking Neural Networks provide.

Having that in mind, we decided to test the developed SNNs in the DVS-Gesture dataset [183]. The gestures were recorded by the iniLabs DVS128 camera which is a 128x128 pixel Dynamic Vision Sensor. This sensor produces a spike at one of the 128x128 pixels when the pixel's value changes over a user-defined threshold. The spike gets encoded as an event where the timestamp of the spike and its spatial coordinates are stored. The consists

of 11 classes, each one representing a gesture. 29 subjects performed each of the gestures in 3 different illumination conditions adding to 957 different elements - collections of events, annotated from 0 to 10. The 11 gestures are the following:

1. Hand clapping
2. Left hand waving
3. Right hand waving
4. Right arm rotation clockwise
5. Right arm rotation counter-clockwise
6. Left arm rotation clockwise
7. Left arm rotation counter-clockwise
8. Forearm roll
9. Air drums
10. Air guitar
11. Other

In fig. 6.1 we see the resulting events produced by the Dynamic Vision Sensor over 2 seconds of one example instance of each class, except the "Other" class.

To load the dataset in the desired format we use the Tonic library in python (https://github.com/neuromorphs/tonic). The library separates the dataset into a train set and a test set, formulated by the gestures of 18 and 11 subjects respectively in all illumination conditions. These sets are objects of Tonic implemented classes and each gesture is a tensor that contains all the events in the following format: Timestamp of the event - X axis - Y axis - polarity of the event. The polarity is represented as a binary value where 0 corresponds to a "negative" spike and 1 to a positive spike. Polarity indicates if the spike was produced due to an increase or decrease of the corresponding pixel's value. The addition of the polarity information slightly transforms the dataset. In fig. 6.2 green pixels indicate the positive spike while the red

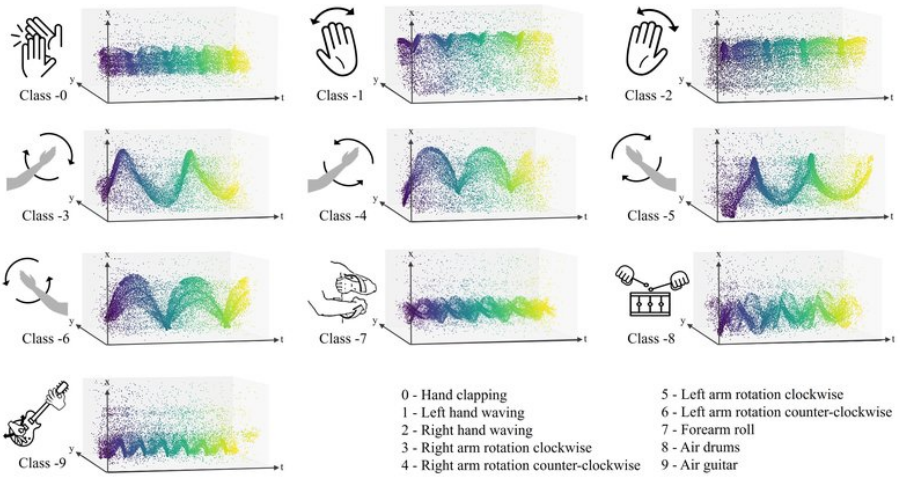



Figure 7.1: An example of events produced by the sensor over 2 seconds of recording. Events get stacked in the time dimension with the yellow-colored represent the first spikes while purple represent the last ones.

ones the negative spike, overall adding to the input information, since the network could potentially use the polarity feature to extract information in the time dimension.

The use of the Tonic library also provides important transformations over the dataset. The most noteworthy of the transformations implemented are the Denoising and Downsample timestamps and/or spatial coordinates. Denoising is used to filter events that are probably caused by noise. This transformation drops events that have no spatial neighbour within a user-defined time duration, clearing the input. Downsampling may be the most important one. Spatial downsample may be used to decrease the size of the input since the original data require $128 * 128 = 16384$ input neurons, and a large enough memory space in processing. Time downsampling results in timestamps of the events being multiplied by a value less than 1, causing the events to be fed to the network faster, and so past information to not be lost in time. It should be noticed that time downsampling does not drop any events.

The Tonic library also implements transformations to deal with the robustness of the network, to tease the data in order for the network to focus and learn general details of the dataset. Such transformations include a random drop of events, flipping spatial and temporal dimensions, reversing the

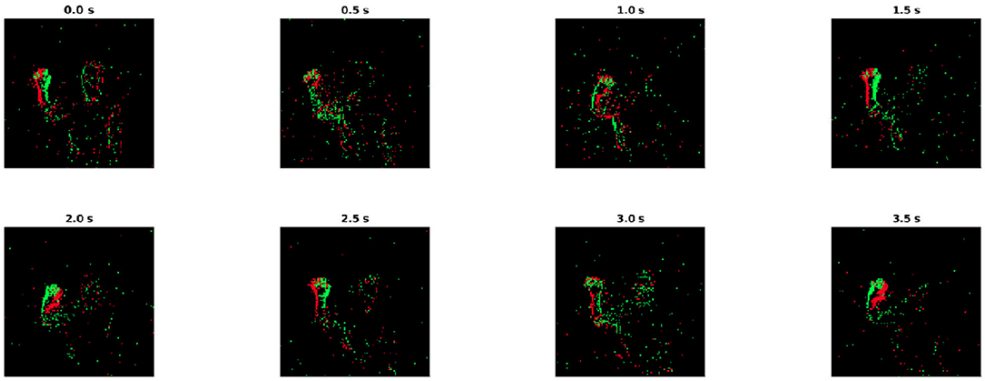



Figure 7.2: Right hand wave instances separated by 0.5 seconds each. Each non-black pixel represents a spike produced by the sensor. Green-colored ones indicate an increase in the pixel's value over the threshold while red ones indicate a decrease of the pixel's value at the given instance.

time, flipping polarities, jittering events spatiotemporally and more, which can be very handful in experiments.

## 7.2 Training the SNN

Having the dataset ready, we decided to focus our attention on the most novel methods of training Spiking Neural Networks. As a result, we chose three methods of training, Back Propagation Through Time (BPTT) a method that has been used in the past for SNNs, as a benchmark, Decolle (ch. 4.3.5), a learning rule suitable for feedforward SNNs and e-prop (ch. 4.3.7), a method used mostly in recurrent SNNs. The choice of these 2 methods over the ones presented in this dissertation or exist in the research field in general, is based on the novelty of the methods and their latest development, being published in 2020 and 2019 respectively. Furthermore, the e-prop method, while it has been tested on solving temporal assignments, it has not faced such complex, in the temporal dimension, datasets yet. Decolle has been tested before in the same dataset, but in this work, we experiment with the size of the network, and other hyperparameters in order to gather more information about the learning rule and compare it with its competitors.

## 7.3 BPTT Training

Back Propagation Through Time (BPTT) is a commonly used learning scheme for recurrent neural networks. Because of the temporal dimension of the data of spiking neural networks, BPTT algorithms, such as the one developed here [184], are also used in training SNNs, either feedforward or recurrent networks. As a result, we decided to train an SNN with BPTT so as to use it as benchmark and compare the results with the newer methods that are tested in this dissertation.

### 7.3.1 Setup

The development of the neural network and its training was done using the Norse library in python [185]. Norse is a library built for developing Spiking Neural Network and gives the freedom of the choice in the network structure such as the elements of the networks e.g. the neurons (IF, LIF, Izhikevich and more neurons have been implemented), the encoding schemes if needed (not necessary for our purpose) providing a framework for an easy way of developing SNNs and controlling its hyperparameters. Norse expands PyTorch with the aforementioned features. Most importantly, it can handle Tonic-loaded datasets. This helps us to load the dataset and apply to it the necessary transformations to manage it and form it in the desired way for our test purposes.

In order to extract important information, multiple networks are developed, teasing one or more parameters of each one. The implemented transformations of Tonic and the easy use of these data from Norse allowed us to experiment with both the neurons' parameters like voltage threshold and spatiotemporal manipulations on the data for better performance in accuracy and speed of the test runs.

### 7.3.2 Network variations

The networks developed to test the BPTT were recurrent SNNs. We experiment with different sized networks and since the transformation of the data was done by Tonic, we tried a variety of them applied to the dataset in order to find out the best values for the given task. Structural elements of the network, such as the neurons, were also altered to gain information about

their response and the performance of the network. Overall, we focused on three main aspects of optimization:

- Preprocessing Optimization
- Network's Architecture Optimization
- Training Optimization

Since the data taken from the sensor, because of its high rate, were too big to either train a network or even fit in the memory, we used spatiotemporal transformations to reduce the data size. We compressed the events by a factor in the range of 1000 to 100000 which is the same as saying that the network manipulates information every 1 to 100 milliseconds (since the frame rate of the sensor is about 100 kHz). This also causes the events fed to the network to be closer in time preserving their temporal correlations without letting their effect fade out in time. Over-downsampling in the temporal dimension though, might cause too many events to be packed together in one input frame - timestamp and the network to be unable to extract any meaningful information. This is why we experiment with multiple downsampling, to achieve the best performance with reasonable speed.

Equally important is the downsampling of the spatial dimension as well. The 128x128 input of the sensor gives a big enough input layer of neurons, being expensive in terms of memory capacity, and also it causes events to be sparse in the spatial dimension. This allows us to reduce the input size downsampling the data and saving memory and time without significant effect on the performance (with the possibility of increasing the performance with downsampling). To see the effect of downsampling of the input data to the network, we test networks with input sizes 128x128, 64x64 and even 32x32 and compare the results.

The structural elements of the network, the neurons, are the ones that define the network and its performance. Equipped with the information gained from the previous analysis on the neuron dynamics, we also decided to tease hyperparameters that affect their response. The most notable and important ones, that were tested in the following experiments, are the voltage threshold and the membrane and synapse decay constants. The voltage threshold was tested in values of 1 and 1.5 to examine the difference in the network dynamics on such occasions. The decay constants were teased to study the network's behavior when the neuron tends to keep the membrane potential

(acting more like an Integrate-and-Fire neuron) and when the immediate feed is crucial for the neuron to spike at all.

In total, to extract the most information in satisfactory training time, since the Norse library was not optimized for such sized tasks (having speed issues compared to the optimized DECOLLE), we implemented 10 recurrent spiking neural networks, where their hyperparameters are such set, so as to give us the ability to extract information. We also applied some modifications such as adding a drop probability factor (where events are randomly ignored, to increase robustness) and changing the alpha factor, a value that has to do with surrogate gradient computation. Although the whole search space for the best setup is not explored, we manage to get meaningful results from the test runs.

### 7.3.3 Results

The preprocessing of the dataset seems to affect the network significantly. In fig. 6.3 and fig. 6.4 we see two identical networks (or could be thought of as the same network) to process the input dataset with temporal downsampling of 100000 and 1000000 respectively. As we see, the increased downsample in the temporal dimension results in worse performance. Temporal downsampling causes events to be packed together in the same timestamp causing the input at each timestamp to be overwhelmed with events that actually happen in long distances through time, resulting in information loss and worse performance of the network. On the other hand, decreasing the temporal downsampling (increasing the multiplying factor) also results in bad performance (fig 6.11). While this seems counterintuitive, it can be explained by the fact that the events may be too sparse in time and their correlation between the temporal dimension to be lost due to a fast decay of the neurons.

Another type of downsampling that has been applied is spatial downsampling. The results show that no or low spatial downsampling (leaving the input size in the tens of thousands) causes high instability and chaotic behavior (fig 6.5). However, it can be seen that the network is increasing its performance through the epochs. This may not be so obvious by the accuracy plot, but it can be seen when we plot the total loss of the network for each epoch fig. 6.3 - where spatial downsampling of factor 0.25 was used. The resulting decreasing total loss in each epoch points out that the network does indeed learn. However, when we compare the performance compared to fig. 6.5, the increase in the test loss of 6.3 (or the decrease of its accuracy) in

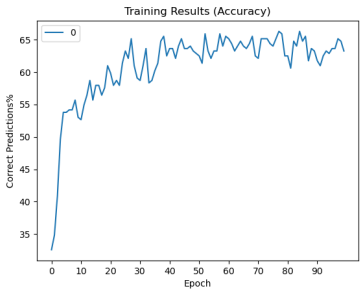



Figure 7.3: Network with temporal downsampling $10^5$


Figure 7.4: Network with temporal downsampling $10^6$

the final epochs makes the choice of such setup unattractive, supported by the fact that at the final epochs both of the networks (fig 6.5, fig. 6.3) have achieved similar results, although further training should be done for more conclusive results. This unstable behavior of the network in this setup may be because of the sparsity of the events in the input space. A huge input size in such a small hidden layer, consisting only of 50 neurons, may cause instability since the hidden layer is unable to distinguish patterns from such a large input, or maybe needed more training time for the pattern recognition to be achieved.


Figure 7.5: No spatial downsampling applied compared to the 0.25 of fig. 6.3

In fig. 6.6 the same network is given the dataset with spatial and temporal downsampling of 0.25 and 100000 respectively but a drop event probability of 0.25, meaning that randomly a quarter of the events will be dropped (discarded), was applied. The resulting training presents lower accuracy

in the final epoch and in general slower learning slope in the first epochs. Nonetheless, it shows a general increase in accuracy for the last epochs that the test ran without the drop probability misses providing information, which may mean that the network could reach higher accuracy scores with more training. These behaviors could be explained with the logic that dropping events results in a more sparse inner state - number of spikes - and the neurons can extract features with greater ease and have a more stable learning curve. The dropout factor can also explain the lower slope learning in the beginning, since dropping these events loses information for the network.


Figure 7.6: Results of applying drop probability to the network

A counterintuitive result from the test run that was done was the fact that the hidden layer size does not seem to affect the performance of the network at all. In fig. 6.3 and 6.7 we see the same setup for 2 networks with the only difference being the size of the hidden layer. The one in fig. 6.3 has 50 hidden layers, while the one shown in fig. 6.7 has 100. However, the performance of the networks is similar, both in the learning slope and in their final accuracy or even stability. This proves the great capabilities of spiking neural networks since only 50 or even fewer neurons perform with around 70 percent accuracy in such a difficult spatiotemporal task.

The instability presented by the network due to the absence of spatial downsampling seems to be widely decreased or even disappear with an adjustment of the learning rate. In fig. 6.8 we reduce the learning rate from the initial 0.002 that resulted in fig. 6.4 to 0.0002 (ten times less). As we can see from the plots, the accuracy steadily increases with no sudden jumps and no large differences in accuracy between epochs (which represents stability). It seems that the lower learning network gives the ability the network to fine tune the large number of weights that connect the input with the hidden


Figure 7.7: Accuracy of network with double neurons in the hidden layer compared to the one of fig. 6.3

layer, with no large jumps in the search space that cause the instability we see in fig. 6.4


Figure 7.8: Results of training with decreased learning rate compared to the one of fig. 6.4

Differences in the variable alpha do not seem to affect the performance of the network. In fig. 6.9 the network has the same well performing spatiotemporal parameters but is divided by 2 the alpha value. The only information that we may extract is that this decrease of the parameter increased the instability and accuracy differences between consecutive epochs. This behavior was expected since the inverse of the alpha value is related with the square root of the gradient calculation with the method of surrogate gradient, and so its decrease results in the increase of the grad value.

The results of the experiment tell us about the result of performance through changes in the characteristics of the neurons. It should be noted

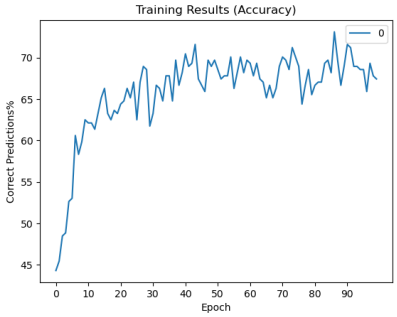



Figure 7.9: Network with alpha value adaptation

that the following test runs were done with temporal downsampling of 10000, which as we saw performs worse due to the sparsity in time of the input data. However, suitable variables can give acceptable performances. As an example the networks in fig. 6.10 and fig. 6.11 differ in the synapse inverse and membrane inverse constants with the pairs 100000, 100000 and 1000, 1000 respectively. The synapse constant controls the refractory period of the neuron, that is the time it needs to come back to its resting state after it emits a spike, while the membrane constant controls the decay of the neuron as it charges with input spikes before it reaches the threshold. With the lower temporal downsampling, we can justify the better performance of the network of fig. 6.11 over the one of fig. 6.10 since lower inverse constant values make the neurons act more like Integrate and Fire (IF) neurons, keeping the received spike longer through time and giving the ability for the temporal information to be present and extracted. The larger values of these parameters result in the neurons decaying faster than a new spike arrives (in combination with the lower temporal downsampling) causing the temporal information to be lost and some neurons to reach the resting state before the next spike arrives. While in this experiment, the runs do not seem to show an improvement in the result of the network's performance, finetuning these parameters may result in higher accuracy.

With the same temporal downsampling factor, we tested the effect of the voltage threshold in the network. The network seems to show worse behavior for an increased voltage threshold. The one shown in fig. 6.12 has the threshold set to 1.5 compared to 1.0 which was the default for the previous networks. The network shows an unstable behavior shows both in its accuracy graph but mostly in its test loss fig. 6.12. This behavior may

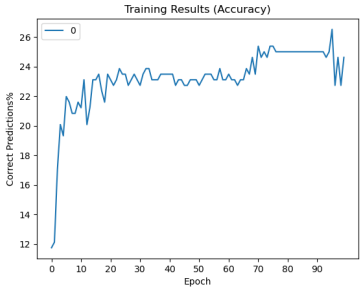
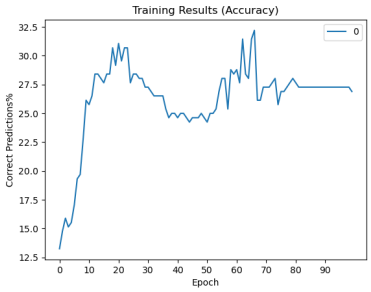
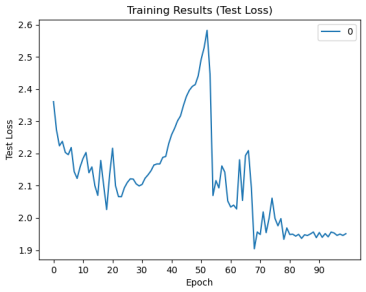



Figure 7.10: Results of increasing membrane inverse constants


Figure 7.11: Results of decreasing membrane inverse constants




(a) Accuracy




(b) Test loss

Figure 7.12: Results of threshold increase compared to the network of fig. 6.3

be explained by the fact that a neuron with a higher firing threshold needs more spikes as inputs, and the weight changes did not manage to adapt to this change. The low temporal downsample may have prevented this necessity and so the network performs as such.

In conclusion, preprocessing of the data seems to play a crucial role in the performance of the network with the most important one being the temporal downsampling. For this particular dataset, the value of 100000 seems to perform best. Spatial downsampling of value 0.25 shows better results than larger values, even though a correct combination of spatial downsampling and learning rate seems to address the stability issues and perform well. This experiment shows that dropping random events does not affect notably the final (maximum) accuracy, while it may increase it, but it does increase

the number of needed epochs to stabilize to the maximum accuracy. Finally, through this experiment, is shown that the neuron's time constants should be calibrated to the input dataset temporal configuration. If events are far apart in time, a more IF behavior is desired for the neuron. Otherwise, a faster decaying and refractory period neuron is required. The voltage threshold should also be adjusted to the temporal density of the events in the dataset since it can affect the ease of spiking in the hidden layer which needs to be either increased in sparse event datasets or decreased in dense ones.

## 7.4 Decolle Training

DECOLLE training [156] enables online training by focusing on local losses that can be computed without the need of the network to calculate the error of its output and propagate it back through the rest of the layers. From the engineering perspective, it is intriguing to note the results of such a scheme as its online learning capability promises high speeds. The measure of the accuracy in such hard tasks will highlight the capabilities of this learning scheme.

### 7.4.1 Setup

Since Decolle has already been tested in the same dataset by the authors, we focused on testing different variations of the network so as to take full advantage of the potential of the learning method and find the suitable network parameters, discovering in the process patterns about the performance over specific hyperparameters. The authors had already implemented the preprocessing of the data and modified them in the format suitable for maximum performance through the training. On this occasion, we did not use the Tonic library to load the data since the needed code already existed. Furthermore, the transformations applies to the other training methods to accelerate the training time were not needed in this case since this dataset, being preprocessed specifically for this task, let the training be fast enough. Moreover, we should notice that the data, while not being transformed by us, already had spatial downsampling from the implemented preprocessing process. The spatial dimension was divided by 4 in each axis resulting in a 32x32 input image.

### 7.4.2 Variations

In this experiment, we focused on finding the ideal hyperparameters of the network. The optimizer that is used is the Adamax optimizer which is well studied and thus we will not focus on the optimization of Adamax. As a result, we trained 14 different networks for 200 epochs over the following parameters:

- Number and type of layer: We use combinations of MLPs and spiking CNNs to achieve deep learning
- Kernel size for convolutional layers
- Neurons' constants such as refractory period of the neuron
- Training chunk size: The training data size being too large from the perspective of GPU memory and training time, it was manually set to a fixed value and each data object was split into chunks of the defined size.
- Learning rate drop factor: determines the drop of the learning rate as the training continues
- Regularization terms: This parameter controls the spiking frequency of the neurons, keeping it (depending on its value) in reasonable values - not letting the neurons become silent or fire constantly. The regularization values may differ in each layer of the network.

The choice of the above hyperparameters was made since we thought that they play a crucial role in the network's performance. The creators of the learning rule provide results only from one network - not mentioning tests being done in other network architectures. Their network was composed of three convolutional layers with kernel size 5. Wanting to explore the possibilities, we focused on deep neural networks with combinations of perceptrons and convolutional layers. We decided on 4 different network architectures that may present interesting results. In the following table 6.1, we show the networks' architecture. We should notice that convolutional layers were always first in the network, before the MLP layers in our implementation.

With the proposed hyperparameters to alter to test the network's performance, the total number of networks to be tested is exponential. To reduce

| ***Network ID*** | ***Conv. layers*** | ***MLP layers*** |
|---|---|---|
| 1 | 64-128-128 | - |
| 2 | 128-512-256 | 128 |
| 3 | 128-512-256 | 128-128 |
| 4 | - | 512-256-128-128 |

Table 7.1: Developed networks' architecture. The order the layers are in start from left to right.

the computational time of this task and still gather the information that will lead us to useful conclusions, we use pair of networks that differ in one element - parameter. This way, we can evaluate the effect that this change will have on this particular setup and probably in general in the whole set of networks. Although we realize that this method will not provide us with definite results, we manage to reduce the set of networks to be tested down to just 14. Furthermore, the information gained from the differences in networks pairs will point to the right direction for the global ideal choice of hyperparameters.

As a starting point, we used the values of the parameters the authors of DECOLLE proposed. Based on these values - Kernel size = 7, alpha = 0.97, alpharp = 0.65, chunk size = 500, drop factor = 5, regularization = 0 - and the networks' architectures developed, we create the following pairs with slight differences to obtain information about the changes in the targeted values. We tested the first network with chunk size 500 and 1500 to see the difference in performance the input time duration chunk causes, since the authors had already worked with the same architecture, probably teasing its hyperparameters. The second network was focused on the effect of regularization. One variation ran with the proposed hyperparameters, while in another one we set the regularization as 0.1 for each of the layers. In two more versions, we dropped the value of deltat, the variable that determines the difference in time over two consecutive timesteps, from 1000 to 500. One of those two versions was tested with regularization 0 and the other had a decreasing format [0.2, 0.15, 0.1, 0.05, 0] for each layer, with the idea to control the input spikes so as to prevent them from continuous spiking and information distortion. Four variations of the third network (ID = 3) were also implemented. The first one was the presented architecture with the proposed parameters. The second had different neurons' hyperparameters.

The membranes' potential decay was set to 0.99 instead of 0.97 and the refractory period decay from 0.65 to 0.55 to keep the neurons more active to contrast the sparsity of the data. The other two versions were tested with kernel size 3 for the convolutional layers and the drop factor was set 5 and 10 respectively to each network. This variable divides the learning rate after a user-defined number of epochs, here 30. The fourth architecture was tested with the parameters set by the rule's authors and was compared with a copy that uses the Adam optimizer. Moreover, 2 final versions of the same architecture were implemented, where they both have an increasing regularization for the layers with the idea to not let the last layers become silent and lose information for the training, with the difference that one of these had 0 regularization to the last layer in order to not affect the final - output layer with spikes not produced by the network. The regularization values for the layers of each network where the following: [0, 0.04, 0.08, 0.12, 0.16] and [0, 0.05, 0.1, 0.15, 0].

### 7.4.3 Results

Analyzing the results from the test runs we notice that the networks trained with Decolle reach high accuracy from early epochs. Compared with the rest of the learning methods tested in this dissertation, this performance difference may be because of the fact that Decolle learning scheme's code was built in such a way to handle solely such datasets. The authors, as stated before, manipulate the dataset in such a way to run in their developed network scheme, losing on universality but aiming for better performance like the one we see from this experiment.

Generally, from the four families of the developed networks (the four different architectures), we notice different behaviors. Networks composed with three convolutional layers fig. 6.13 fig. 6.14, the first layers seem to struggle to learn their local learning rules. The first layer tends to stay at around 25 percent accuracy throughout the learning process, while the rest of the layers proportionally increase their accuracy. At the last epochs, it seems that the output layer scores around the same accuracy as the last convolutional one. However, in contrast with the rest of the families, the output layer for the first epochs score lower than layer 3 and it seems that it needs a few epochs to catch up. A possible explanation could be that convolutional layers focus on feature extraction prior to the local learning rule (and for this reason, they score lower) and the output layer, as the only

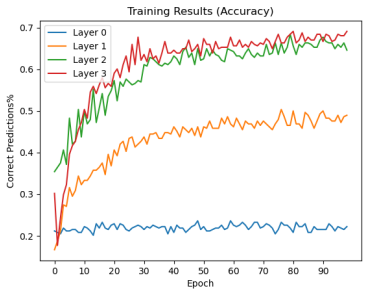



Figure 7.13: Network composed with three convolutional layers

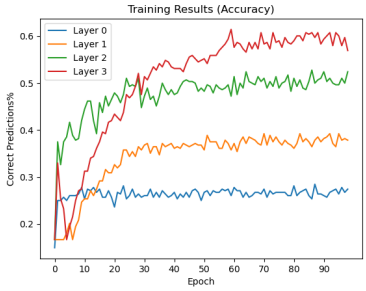



Figure 7.14: Network composed with three convolutional layers, with increased chunk size

perceptron layer needs some time to learn the features from the convolutional layer 3, and so it catches up later.

Networks with 3 convolutional and 1 perceptron layer show the best behavior fig. 6.15, fig 6.16. With the exception of the first layer, the rest seem to reach accuracies around 90 percent. However, the first layer compared to the first family seems to learn from the local rules reaching accuracy over 50 percent. Layer 2 seems to need some epochs to catch up but eventually reaches the accuracy scores of the last layers. Another important observation is the stability the network presents through the epochs, that is, differences in accuracy between consecutive epochs stay low.

When we added another perceptron layer to the network (three convolutional and 2 perceptron layers), the network did not improve fig. 6.17, fig.6.18. We observe an increased instability throughout the layers compared to the last setup. Moreover, adding the perceptron layer seems to make the performance of the second layer drop. Generally, we may deduce that only the three last layers of the network (output and the last 2 hidden ones) may reach close accuracy. Since the accuracy of the output did not increase, and in some cases decreased, adding too many layers may cause instability and performance issues.

The fourth architecture that contains only 4 perceptron layers presented a different behavior compared to the rest fig. 6.19, fig. 6.20. These networks scored lower than networks with both convolutional and perceptron layers, reaching an accuracy of less than 90 percent. Moreover, they present a

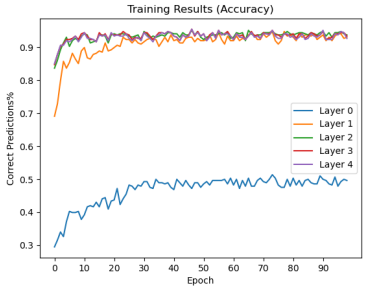



Figure 7.15: Network composed with three convolutional layers and one perceptron layer with deltat parameter set to 1000


Figure 7.16: Network composed with three convolutional layers and one perceptron layer with deltat parameter set to 500

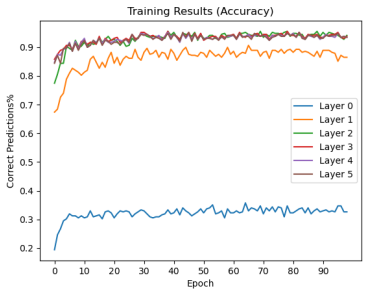



Figure 7.17: Network composed with three convolutional layers and 2 perceptron layers with deltat parameter set to 1000


Figure 7.18: Network composed with three convolutional layers and 2 perceptron layers with deltat parameter set to 200

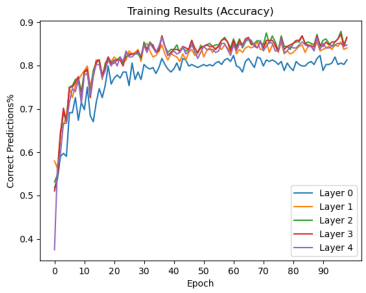



Figure 7.19: Network composed with 4 perceptron layers

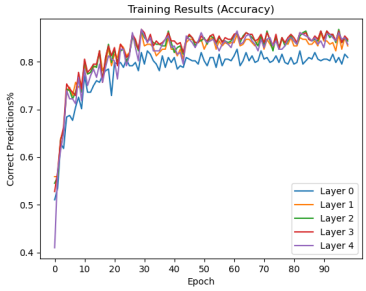



Figure 7.20: Network composed with 4 perceptron layers with regularization applied

more unstable behavior with significant fluctuations in accuracy between epochs. However, a noteworthy observation is the accuracy of the first layer. In contrast to the other families of networks, where the first layer could not exceed 60 percent accuracy, here we observe the accuracy of layer one to follow closely the rest layers, reaching values over 80 percent fig. 6.19. Finally, we can notice that all layers score high even from the first epoch (most of the time over 50 percent for every layer). The increased accuracy of the first perceptron layer may be due to the focus on the local learning rule compared to the feature extraction that convolutional layers perform.

Testing in increasing the time length of the input sequence in the network (with the idea that the network receives a larger chunk and may learn better with each application of the learning) showed that this manipulation causes a drop in performance fig. 6.13, fig. 6.14. This indicates the inability of the learning scheme to deal with large time sequences as the authors themselves proposed. This weakness may be the result of the lack of any neuron trace to inform later neurons about events in the more distant past.

Manipulating the learning rate and its drop rate seems to have a great impact on the performance of the network. Comparing two identical networks (fig. 6.19, fig. 6.21) with different learning rates - $10^{-9}$ for the left and $10^{-8}$ for the right one - we notice that increasing the learning rate caused the network to become unstable and reach lower accuracy overall. This can be explained due to the large jumps in the search space resulting in the inability of the network to reach a minimum. The same behavior is observed when 2 networks with different learning rate drop factors are compared fig. 6.18,

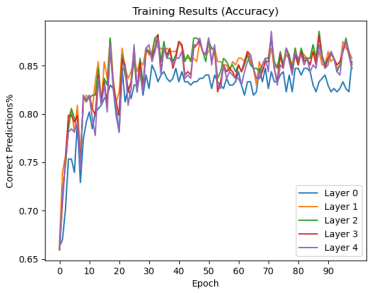



Figure 7.21: Increased learning rated compared to fig. 6.19


Figure 7.22: Decrease of LRDF of the network of fig. 6.18

fig. 6.22. The learning rate drop factor (LRDF) divides the learning rate with the parameter's value every 30 epochs (the interval of 30 epochs was kept the same for all the networks since it seems reasonable this division interval for the number of epochs chosen). The decreased (LRDF) caused the network fig. 6.22 to become less accurate. The cause of this could be that in later epochs that the network should be finetuning (after the general features have been learned), the learning rate is still high and the network makes large jumps in the search space, resulting in the inability to reach its local minimum (apply fine tuning).

The addition of regularization in the networks seems to generally stabilize the learning process (fig. 6.19, fig 6.20). The application of increasing regularization in deeper layers did not show to produce different results in the terms of accuracy compared to the decreasing or constant (same for all networks) regularization. However, the regularization values never exceeded 0.2 for our experiment and higher values may affect the network in unpredictable ways.

The experiment also revealed that the choice of the membrane time constant and the refractory period of the neuron affects significantly the performance of the network and small changes result in notable differences. The membrane time constant controls the decay rate of the neuron (how fast it discharges between input spikes) and the refractory period the time it takes the neuron to be ready to handle inputs after it spikes. Fig 6.23 and fig. 6.24 show the same network where fig. 6.23 has the parameters pair as 0.97 and 0.65 while fig. 6.24 is set to 0.99 and 0.55, making the neuron "sharper",


Figure 7.23: Network's performance with stable neurons' behavior


Figure 7.24: Result of increasing the sharpness of the neurons

shortening the time interval of both of the aforementioned actions. As a result of this change, the network performed worse and became more unstable. Moreover, the output layer did not receive as good scores as the middle layers. This may be the result of the neurons forgetting faster events from past times (due to the change in membrane constant) causing a drop in the fire rate and the network having a hard time extracting information and finding a stable configuration for the more unstable neurons (for the given setup).

Analyzing the results of networks with different deltat parameters revealed that its value affects the network's performance. Comparing the networks of fig. 6.15, fig. 6.16 and 6.17, 6.18 a pattern emerges. The decreased value of deltat causes instability to the network and seems to slightly drop its accuracy. Deltat controls the time intervals at which the neurons receive spikes, and in a sense controls the frequency that the network operates at. Setting a low value means we drop the frequency and as a result, lose information. Suitably, the parameter should be fine-tuned with the input frequency, a process done automatically in a neuromorphic platform.

Finally, the tested kernel size values tested 3, 5, 7 did not seem to have any notable effect on the network's performance.

Overall, based on the results of the experiment, we conclude that the best results for the given task are produced by a network consisting of three convolutional and one perceptron layer. The use of only convolutional or perceptron layers achieves worse scores and so does the addition of extra perceptron layers in the proposed architecture. A stable performance requires a reasonably low learning rate and high drop factor to avoid large jumps and

scattering in the search space. The addition of regularization in the layers seems to help the network stabilize while the inner structure of the neuron needs finetuning for each given dataset since it seems to play a major role in the network's performance. One important observation made through the experiment was the effect that the input time window has on the performance. The results lean in the direction of the inability of the learning scheme to process well long time intervals by nature.

## 7.5 E-prop Training

The E-prop learning rule attracts a lot of interest from a neuroscientific perspective since it seems to mimic the behavior of the brain as from the network structure (RSNN) as from the structural elements that form the rule itself i.e. the eligibility trace and the learning signal. As a result, the performance of this learning scheme in such a dataset is of great interest from both engineering and neuroscientific standpoint.

### 7.5.1 Setup

The preprocessing of the dataset, such as the required transformations, was done using the Tonic library mentioned before. The transformations applied to the dataset is the Downsampling of spatial coordinates, from 128x128 to 32x32 so as to shorten the input size. Furthermore, since the events occur with a significant time difference, meaning that a block of 4x4 pixel is improbable to spike at the same time, this downsample can be justified and its effect through the learning process be negligible. However, this sparsity of the events both in the spatial and temporal dimension gives us the ability to downsample the time dimension compacting the events without any loss of information. The scale of downsampling in time is set to 0.001. This way, the learning process is significantly accelerated (the learning process of e-prop is more time-consuming than the typical BPTT as the authors themselves have declared [162].

The software used for the training is Neko [164], a python library for Neuromorphic learning rules, in which the e-prop method has been implemented. The authors have coded the development of an RNN network with a variety of backends such as PyTorch and TensorFlow, learning rules such as BPTT and e-prop and neuron models for the network i.e. the simple recur-

rent cell, the LIF recurrent and the adaptive LIF (A-LIF) recurrent cell. The following experiments use the PyTorch backend and the method of e-prop, since the BPTT has already been tested. However, the algorithm was not coded as such to handle data from the Tonic library. The problem was in the data format of the Tonic objects, where it contains lists of 4 elements, each one representing the Timestamp, the X axis, the Y axis and the polarity respectively, while the training demanded the input to be a tensor of 128x128=16384 trinary values as inputs for each of the neuron of the input layer. The implementation of the above transformation was done for each batch separately using the Numpy library for faster execution. However, it seemed to be of high computational cost. It should be noted that such behavior can be avoided with the use of real-time DVS data or the correct/suitably implemented platform.

### 7.5.2 Variations

As in previous experiments, here we focused on three types of manipulative classes. First, the preprocessing of the data was taken into account so as to achieve an ideal mix of speed and accuracy since, as mentioned before, Neko is not implemented to handle Tonic datasets. Next, the network's architecture was chosen to be tested. In the experiment we developed recurrent SNNs and tested different sizes of the network. Neko implements a useful parameterization in the neurons' type and the regularization of the network's spike firing, which we used in our experiment. While it does not offer neurons' inner characteristics like tau values as parameters (besides Voltage threshold), we tested different values of such parameters by altering the code of functions of the library. Finally, the learning scheme itself gives three modes in connecting the output neurons with the recurrent ones to transfer the learning signal (ch. 4.3.6), which Neko has implemented and that we have used in this experiment.

The way that Neko implements SNNs and E-prop learning results in the need for the whole time length of the input to be fed into the network. While it is well established in theory that the update function of the network and the learning can be done online (not having the need for the whole signal in time), in order for the library to take advantage of PyTorch tensors and GPU acceleration, a tensor that contains all the timesteps is used (even the ones that have no spike produced). This results in a huge need for RAM (a couple of TeraBytes in memory), since the dataset may contain

hundreds of thousand of events per gesture, each one multiplied by 16384 to represent the input size. To overcome this problem, we downsampled the time dimension by 1000, (since lower tested values resulted in memory allocation problems). We did not proceed in spatial downsampling or further temporal downsampling since the dataset was already compressed in its whole size by the factor of a thousand.

E-prop does not require unfolding in time, but as stated before, the Neko framework keeps in memory past events and neurons' states in order to update and produce outputs. This increases the spatial-memory complexity of the algorithm making it, in practice, the same as BPTT. As a result, the possible network size was limited in order to not exceed the memory capacity, while in a suitably implemented e-prop (like in an applied neuromorphic platform) such problems would not occur. We tested networks with sizes of 64, 128, 256, 512 and 1024 neurons before exceeding the memory capacity, for 200 epochs. The neurons' type was chosen to be LIF or adaptive LIF (A-LIF) for the network, a neuron type that the developers coded based on the paper [162]. A-LIF neurons have a hyperparameter, the neuron's adaptivity, which controls the rate the neuron adapts - learns to reach the suitable values. We applied three different values 1.0, 2.0 and 4.0 in order to move to the optimal value. Regarding the neurons' responses, we tested in changing the membrane time constants between the values 0.001, 0.02 and 0.2 and the membrane voltage threshold between 0,5 and 1.0. Voltage threshold was not tested further based on the idea that an input spike of value 1 will cause the neuron to spike for a lower voltage threshold while the network's weights will adapt and end up in proportional values for any membrane voltage threshold difference (i.e. the weights will simply double for a doubled voltage threshold and the network will have the same behavior). Having that in mind, we tested with the two aforementioned values so as to reduce the training size and gather information about whether our conjecture stands or not.

In early experiments we ran, we noticed that the network tends to stabilize in a firing threshold of around 100 spikes per second per neuron. Having that in mind, we tested the effect of the regularization parameter on the network's performance setting it in values 80 and 150 with coefficients 0.00001 and 0.001 respectively. Finally, we applied each implemented e-prop mode for the weights of the learning signal as presented in ch. 4.3.6, which are random, symmetric and adaptive.

Table 6.2 shows the total possible search space of our experiment regarding the e-prop algorithm. However, accessing the whole search space requires

| Parameters | Values | | | | |
|---|---|---|---|---|---|
| Size | 64 | 128 | 256 | 512 | 1024 |
| Firing Threshold | 0.5 | | 1.0 | | |
| E-prop mode | Random | | Symmetric | | Adaptive |
| Regularization target | 80 | | 150 | | |
| Regularization coef. | 0.00001 | | 0.001 | | |
| Membrane time const. | 0.001 | | 0.02 | | 0.2 |

Table 7.2: Parameters optimized and their respective values

the run of 360 different networks. Since such a task would require a monumental amount of time, we work around the problem using the method we applied in Decolle. That is, we compared pairs of networks with altered multiple variables that do not affect each other to gather information in a much smaller search space (i.e. three networks test three different variables and each pair presents information for each variable). Though, as stated before, this method does not guarantee the perfect search (since the non-paired variable may affect in an unforeseen way the performance), it does reduce the search space massively. Using this idea we developed and tested 16 different networks considering this to be a tolerable balance between information gathering and test runs.

### 7.5.3 Results

Running the above configurations of the network on the proposed dataset, we store the information of the test and total loss and the accuracy of the network for each of the 200 epochs, and the true and predicted output of the last test run, as we did in the previous setups (BPTT and Decolle). With this information we plot the respective plots (test-total loss and accuracy) and the confusion matrices and accuracy per class for the final run. With this representation of the gathered information we analyze the produced results and deduce the effect of each hyperparameter on the network and the reason for it.

A quick deduction that can be made from the performance of the networks

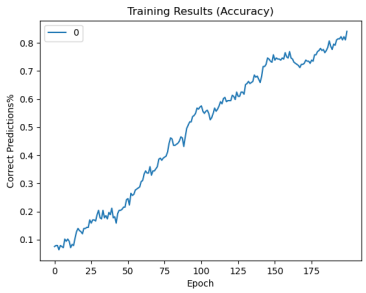



Figure 7.25: Accuracy of a network with 64 neurons in the hidden layer


Figure 7.26: Accuracy of a network with 512 neurons in the hidden layer

is the correlation between the increased number of neurons and the instability of the training. The plots in the following figure present the accuracy of two identical networks with the difference of the hidden layer size. In fig. 6.25 the network has 64 neurons and in fig. 6.26, 512. The pattern follows with networks that have bigger hidden layer sizes perform worse and being unstable than smaller ones. The chaotic behavior may be explained by the inability of the network to extract meaningful information when that many neurons have to cooperate for the given setup and task. Furthermore, E-prop is built in a way that the future effect of a spike on the network is not taken into account for error minimization. A larger network produces more spikes overall (as it is easily understood) and these spikes, while will eventually have an important role in the output that the network will produce, they are not taken into account. This may support that the proposed method might not work well with larger networks or even big temporal data.

The models in fig. 6.25 and 6.26 have been developed with adaptive LIF neurons. Changing the neuron type of the network to LIF produces results such as the one in fig. 6.27. The network in fig. 6.27 uses LIF neurons and it should be noted that it has 512 neurons in the hidden layer. This shows an important result, that is, LIF neurons may present stable results despite the size of the network. This added stability causes the network to have increased accuracy and due to its increased number of neurons, achieve high accuracy from early epochs.

Compared to the bad performance of the network shown in fig. 6.25 and a network with 512 neurons in the hidden layer and a lower adaptation


Figure 7.27: Network with 512 LIF neurons

value for the A-LIF neuron (its accuracy shown in fig. 6.28) shows that the stability issue is probably caused by the adaptation value of the neurons. High values may cause the increased numbers of neurons to overshoot in the process of reaching a global minimum and cause the network to oscillate. As a result, a proposed method could be that a larger hidden layer requires smaller adaptation values for the network.


Figure 7.28: Results of lowering the adaptation values of A-LIF neurons - compared with fig. 6.25

Regarding the neurons' characteristics, through our experiment, the following pattern emerged for the e-prop algorithm. Reducing the membrane constants of the neurons in the hidden and output layers may seem to slightly increase the network's performance. The cause of this effect could be that, with the applied temporal downsampling, the events that are fed to the network are close together in the temporal dimension. This could have the effect that neurons' membrane potential does not discharge fast enough and

the temporal information does not contribute as much. The membrane's constant controls the decay rate of the neuron's membrane potential and its decrease causes a faster decay rate (as the negative inverse is used as an exponent). This could result in the spikes being more sparse in the temporal dimension inside the hidden layer and the network to extract better information resulting in the performance presented. Similarly, the increased value of the membrane's constants, meaning the neuron act more like an Integrate-and-Fire (IF neuron model) may overwhelm the model with spikes resulting in the incapability of the model to learn meaningful patterns as well. However, these effects seem to be minuscule and even changes in orders of magnitude (multiplied or divided by 10) do not play any significant role in the network's performance. Fig. 6.29 shows the accuracy values of such a network with increased neuron's membrane constant, presenting a form of instability, although the network was implemented with A-LIF neurons that may be the cause of this behavior. The applied changes in voltage threshold do not seem to affect in any significant or observable, through this experiment, way, positive or negative, and so the default value of 1.0 could be an acceptable choice.

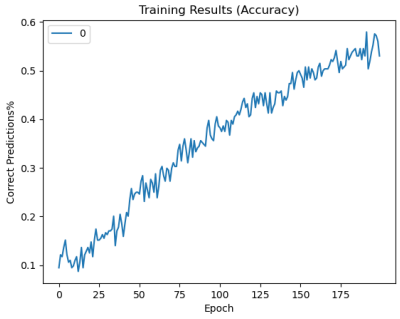



Figure 7.29: Results after increasing time constants of neurons

The choice for the so-called e-prop mode between random, symmetric and adaptive (weights for the learning signal), plays an important role in the network's performance. Adaptive weights seem to follow the pattern of "learning together with the network". Networks that were implemented with this mode used to have a smooth increasing curve regarding the accuracy (fig. 6.28). They also seem to need more epochs until they reach the maximum possible accuracy. The reason for this behavior could be the fact that the adaptive weight needs several epochs to reach the optimal values and help

the network stabilize. Random mode follows the same behavior (fig.6.30) with the difference that it needs more epochs to achieve high accuracy since the weights of the learning signals do not change over time to increase the learning speed. Finally symmetric seem to show the fastest learning ability since the test loss function in fig. 6.31 of an optimized symmetric network reaches a minimum in the first epochs compared to the test loss function of adaptive (6.32) or random (fig 6.33) modes but the increase of its value that we see may be due to an overfit that adaptive mode could potentially avoid. It should be noted that random mode in fig (6.33) also shows a slight increase in the test loss in the last epochs that may be caused by the same reason as stated before, while adaptive mode does not seem to have such behavior.

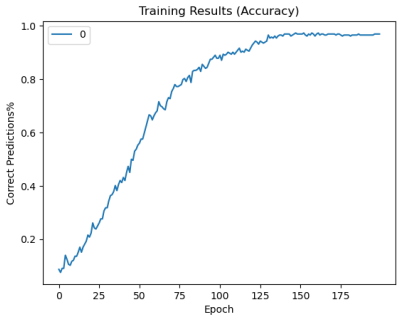



Figure 7.30: Accuracy of network built with random e-prop mode

Summing up, the most important information we extracted from this experiment is the fact that for large-sized networks the e-prop algorithm presents instability issues. The cause of this could be the fact that during learning future influence of the spikes are not taken into account, and large networks, producing an increased number of spikes, are significantly affected. The instability issues could be more dramatic with an unsuitable choice of the adaptation value for the ALIF neuron but carefully chosen values may overcome such problems, while LIF neurons do not present such issues. The choice of e-prop mode is very important and as shown the random performs worse than the other two. Symmetric reaches high accuracy early but seems to suffer from overfitting whereas adaptive, while it needs more epochs to converge to its maximum, seems to overcome the overfit problem through the adaptation of the weights. The inner neurons' characteristics do not seem to affect the network's performance.

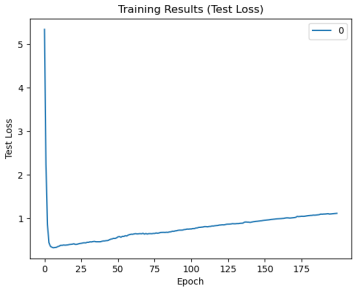



Figure 7.31: Test loss of the symmetric e-prop mode

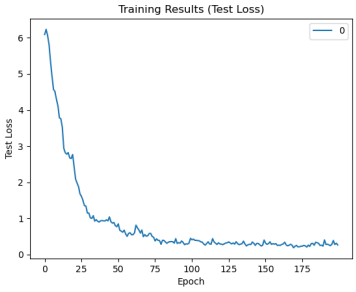



Figure 7.32: Test loss of the adaptive e-prop mode

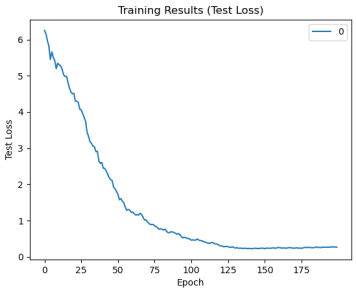



Figure 7.33: Test loss of the random e-prop mode

# Chapter 8

# Conclusion

We have finally reached the final section of our thesis, its conclusion. This dissertation is the culmination of months of research, gathering, editing, and adapting information to approach problems utilizing spiking neural networks and eventually be able to classify data recorded by Dynamic Vision Sensors. To begin, it was necessary to comprehend the fundamental components of the human brain: neurons, synapses, and other structural pieces that unite to build massive network structures that support cognitive processes. It was also required to comprehend the math characterizing neurons (neuron models) and how their spike generating feature enables them to send information via synapses.

Understanding the biology behind spiking neural networks was extremely beneficial in realizing the immense complexity of the human brain, that spiking neural networks are merely a simplification of what evolution has accomplished, and that we still have a long way to go before achieving human level artificial intelligence. However, delving into brain research can lead to new advances in machine learning research by better understanding the systems of the human brain that aid in visual input processing and learning. Neuron assemblies and oscillations, paired with the concept of plasticity, are two essential aspects of the human brain that we found intriguing.

Neuron assemblies improve information processing efficiency, dynamicity, and flexibility, and no research publication has been discovered that uses neuron assemblies for a machine learning task. As previously demonstrated, oscillations can be used to control network activity, synchronize (sensory input from two ears in the owl), and correct neuron activity (saccadic eye movement). In certain publications, oscillations have been utilized to produce

intrinsic network activity for tasks such as spatial navigation and speech [186], as well as in combination with STDP [187] as a more general learning scheme.

As authors of this dissertation, we hope to inspire other academics to delve deeper into brain research in order to find a way to incorporate neuron assemblies and oscillations in machine learning tasks, which could lead to more efficient learning. Furthermore, we felt it was important to learn about existing computing systems as well as current neuromorphic computing research that can be used as a platform to model spiking neural network activity. It was also important to grasp the fundamental operating principles of DVS devices in order to perform the appropriate pre-processing to the dataset. Given our limited amount of time, we decided to focus on and experiment with current learning approaches for classifying gestures using IBM's DVS Gestures dataset.

# Bibliography


[1] T. Nguyen, "Total number of synapses in the adult human neocortex," *Undergraduate Journal of Mathematical Modeling: One + Two*, vol. 3, 5 2013.

[2] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics*. Cambridge University Press, 2014.

[3] D. Balduzzi and G. Tononi, "What can neurons do for their brain? communicate selectivity with bursts," *Theory in Biosciences*, vol. 132, pp. 27–39, 3 2013.

[4] P. Garcia-Lopez, V. Garcia-Marin, and M. Freire, "The histological slides and drawings of cajal," 3 2010.

[5] A. S. Badin, F. Fermani, and S. A. Greenfield, "The features and functions of neuronal assemblies: Possible dependency on mechanisms beyond synaptic transmission," *Frontiers in Neural Circuits*, vol. 10, 1 2017.

[6] Z. Li and M. Sheng, "Some assembly required: The development of neuronal synapses," pp. 833–841, 11 2003.

[7] S. G. Hormuzdi, M. A. Filippov, G. Mitropoulou, H. Monyer, and R. Bruzzone, "Electrical synapses: A dynamic signaling system that shapes the activity of neuronal networks," pp. 113–137, 3 2004.

[8] E. J. Furshpan and D. D. Potter, "Transmission at the giant motor synapses of the crayfish," *The Journal of Physiology*, vol. 145, 3 1959.

[9] G. D. Fischbach, "Synapse formation between dissociated nerve and muscle cells in low density cell cultures," *Developmental Biology*, vol. 28, 6 1972.

[10] A. Peinado, R. Yuste, and L. C. Katz, "Gap junctional communication and the development of local circuits in neocortex," *Cerebral Cortex*, vol. 3, 1993.

[11] R. Yuste, A. Peinado, and L. Katz, "Neuronal domains in developing neocortex," *Science*, vol. 257, 7 1992.

[12] Z. Molnár and K. S. Rockland, "Cortical columns," 2020.

[13] T. Rose and M. Hübener, "Synapses get together for vision," *NATURE*, 2017.

[14] M. London and M. Häusser, "Dendritic computation," pp. 503–532, 2005.

[15] M. Persike and G. Meinhardt, "Contour integration with corners," *Vision Research*, vol. 127, 10 2016.

[16] M. F. Iacaruso, I. T. Gasler, and S. B. Hofer, "Synaptic organization of visual space in primary visual cortex," *Nature*, vol. 547, pp. 449–452, 7 2017.

[17] C. Koch, J. L. Davis, A. B. Book, P. S. Churchland, V. S. Ramachandran, and T. J. Sejnowski, "Large-scale neuronal theories of the brain edited by 2 a critique of pure vision'," 1994.

[18] R. C. Decharms and A. Zador, "Neural representation and the cortical code," pp. 613–647, 2000.

[19] C. Salzman, C. Murasugi, K. Britten, and W. Newsome, "Microstimulation in visual area mt: effects on direction discrimination performance," *The Journal of Neuroscience*, vol. 12, 6 1992.

[20] M. J. Tovee, E. T. Rolls, A. Treves, and R. P. Bellis, "Information encoding and the responses of single neurons in the primate temporal visual cortex," *Journal of Neurophysiology*, vol. 70, 8 1993.

[21] W. Bair and C. Koch, "Temporal precision of spike trains in extrastriate cortex of the behaving macaque monkey," *Neural Computation*, vol. 8, 8 1996.

[22] G. T. Buracas, A. M. Zador, M. R. DeWeese, and T. D. Albright, "Efficient discrimination of temporal patterns by motion-sensitive neurons in primate visual cortex," *Neuron*, vol. 20, 5 1998.

[23] M. Rucci, E. Ahissar, and D. Burr, "Temporal coding of visual space," pp. 883–895, 10 2018.

[24] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *NATURE*, vol. 381, pp. 520–522, 6 1996.

[25] W. Bialek, F. Rieke, R. R. D. R. V. Steveninck, and D. Warland, "Reading a neural code," *Science*, vol. 252, pp. 1854–1857, 1991.

[26] R. Lestienne, "Determination of the precision of spike timing in the visual cortex of anaesthetised cats," *Biological Cybernetics*, vol. 74, 1 1996.

[27] Y. Sakurai, "How do cell assemblies encode information in the brain?" *Neuroscience  Biobehavioral Reviews*, vol. 23, 10 1999.

[28] G. Gerstein, D. Perkel, and K. Subramanian, "Neuronal assemblies," *IEEE Trans Biomed Eng.*

[29] W. Singer, "The formation of cooperative cell assemblies in the visual cortex," p. 7, 1990.

[30] C. J. Shatz, "The developing brain," *Scientific American*, vol. 267, 9 1992.

[31] H. Markram, P. J. Helm, and B. Sakmann, "Dendritic calcium transients evoked by single back-propagating action potentials in rat neocortical pyramidal neurons." *The Journal of Physiology*, vol. 485, 5 1995.

[32] H. Markram, "Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps," *Science*, vol. 275, 1 1997.

[33] W. Gerstner, R. Kempter, J. L. van Hemmen, and H. Wagner, "A neuronal learning rule for sub-millisecond temporal coding," *Nature*, vol. 383, 9 1996.

[34] H. Markram, W. Gerstner, and P. J. Sjöström, "Spike-timing-dependent plasticity: A comprehensive overview," *Frontiers in Synaptic Neuroscience*, vol. 4, 2012.

[35] E. Başar, C. Başar-Eroğlu, S. Karakaş, and M. Schürmann, "Brain oscillations in perception and memory," *International Journal of Psychophysiology*, vol. 35, pp. 95–124, 3 2000. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167876099000471

[36] W. Singer and C. M. Gray, "Visual feature integration and the temporal correlation hypothesis," *Annual Review of Neuroscience*, vol. 18, pp. 555–586, 3 1995. [Online]. Available: http://www.annualreviews.org/doi/10.1146/annurev.ne.18.030195.003011

[37] W. Singer, "Neuronal synchrony: A versatile code for the definition of relations?" *Neuron*, vol. 24, pp. 49–65, 9 1999. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0896627300808211

[38] S. Furber, "Large-scale neuromorphic computing systems," *Journal of Neural Engineering*, vol. 13, 10 2016.

[39] T. Clayton, K. Cameron, B. R. Rae, R. K. Henderson, A. Murray, N. Sabatier, G. Leng, and E. Charbon, "An implementation of a spike-response model with escape noise using an avalanche diode," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 5, pp. 231–243, 2011.

[40] R. J. Baker, S. K. Tewksbury, and J. E. Brewer, "Cmos circuit design, layout, and simulation second edition," 2005. [Online]. Available: www.copyright.com.

[41] H. Nair, J. P. Shen, and J. E. Smith, "Direct cmos implementation of neuromorphic temporal neural networks for sensory processing," 8 2020. [Online]. Available: http://arxiv.org/abs/2009.00457

[42] L. . Chua, "Memristor-the missing circuit element," p. 507, 1971.

[43] H. Wang, H. Li, and R. E. Pino, "Memristor-based synapse design and training scheme for neuromorphic computing architecture."

[44] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, pp. 1297–1301, 4 2010.

[45] I. Boybat, M. L. Gallo, S. R. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, and E. Eleftheriou, "Neuromorphic computing with multi-memristive synapses," *Nature Communications*, vol. 9, 12 2018.

[46] I. Žutić, J. Fabian, and S. D. Sarma, "Spintronics: Fundamentals and applications."

[47] X. Wang, Y. Chen, H. Xi, H. Li, and D. Dimitrov, "Spintronic memristor through spin-thorque-induced magnetization motion," *IEEE Electron Device Letters*, vol. 30, pp. 294–297, 2009.

[48] J. Torrejon, M. Riou, F. A. Araujo, S. Tsunegi, G. Khalsa, D. Querlioz, P. Bortolotti, V. Cros, K. Yakushiji, A. Fukushima, H. Kubota, S. Yuasa, M. D. Stiles, and J. Grollier, "Neuromorphic computing with nanoscale spintronic oscillators," *Nature*, vol. 547, pp. 428–431, 7 2017.

[49] J. Grollier, D. Querlioz, K. Y. Camsari, K. Everschor-Sitte, S. Fukami, and M. D. Stiles, "Neuromorphic spintronics," pp. 360–370, 7 2020.

[50] K. Moon, S. Lim, J. Park, C. Sung, S. Oh, J. Woo, J. Lee, and H. Hwang, "Rram-based synapse devices for neuromorphic systems," *Faraday Discussions*, vol. 213, pp. 421–451, 2019.

[51] J. H. Cha, S. Y. Yang, J. Oh, S. Choi, S. Park, B. C. Jang, W. Ahn, and S. Y. Choi, "Conductive-bridging random-access memories for emerging neuromorphic computing," pp. 14 339–14 368, 7 2020.

[52] J. Park, "Neuromorphic computing using emerging synaptic devices: A retrospective summary and an outlook," pp. 1–16, 9 2020.

[53] M. Suri, V. Sousa, L. Perniola, D. Vuillaume, and B. DeSalvo, "Phase change memory for synaptic plasticity application in neuromorphic systems," 2011, pp. 619–624.

[54] P. A. Merolla, J. V. Arthur, B. E. Shi, and K. A. Boahen, "Expandable networks for neuromorphic chips," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, pp. 301–311, 2007.

[55] N. Akbari and M. Modarressi, "A high-performance network-on-chip topology for neuromorphic architectures," vol. 2. Institute of Electrical and Electronics Engineers Inc., 8 2017, pp. 9–16.

[56] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B.-D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores." IEEE, 8 2013.

[57] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, 5 2014.

[58] L. A. Plana, D. Clark, S. Davidson, S. Furber, J. Garside, E. Painkras, J. Pepper, S. Temple, and J. Bainbridge, "Spinnaker," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 7, 12 2011.

[59] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, 8 2013.

[60] B. J. Shastri, A. N. Tait, T. F. de Lima, M. A. Nahmias, H.-T. Peng, and P. R. Prucnal, "Neuromorphic photonics, principles of neuromorphic photonics," 2018.

[61] M. G. Johnson and S. Chartier, "Spike neural models (part i): The hodgkin-huxley model," *The Quantitative Methods for Psychology*, vol. 13, pp. 105–119, 5 2017.

[62] M. Nelson and J. Rinzel, "The hodgkin-huxley model."

[63] by David Beeman, "Supplementary material for bmm218: Introduction to realistic neural modeling and bmm220: Genesis modeling tutorial genesis resources installation guidelines datasheet genesis resources," *GENESIS Modeling Tutorial. Brains, Minds and Media*, vol. 1, p. 220, 2005. [Online]. Available: http://www.dipp.nrw.de/lizenzen/dppl/dppl/DPPL$_v$2$_e$n$_0$6 − 2004.*html*.

[64] T. Levi, F. Khoyratee, S. Saïghi, and Y. Ikeuchi, "Digital implementation of hodgkin–huxley neuron model for neurological diseases studies," *Artificial Life and Robotics*, vol. 23, pp. 10–14, 3 2018.

[65] H. Yedjour, B. Meftah, O. Lézoray, and A. Benyettou, "Edge detection based on hodgkin–huxley neuron model simulation," *Cognitive Processing*, vol. 18, pp. 315–323, 8 2017.

[66] A. Amirsoleimani, M. Ahmadi, A. Ahmadi, and M. Boukadoum, "Brain-inspired pattern classification with memristive neural network using the hodgkin-huxley neuron."

[67] A. F. Strassberg and L. J. Defelice, "Article limitations of the hodgkin-huxley formalism: Effects of single channel kinetics on transmembrane voltage dynamics."

[68] W. M. Kistler, W. Gerstner, and J. L. V. Hemmen, "Reduction of the hodgkin-huxley equations to a single-variable threshold model." [Online]. Available: http://direct.mit.edu/neco/article-pdf/9/5/1015/813679/neco.1997.9.5.1015.pdf

[69] N. Brunel and M. C. V. Rossum, "Quantitative investigations of electrical nerve excitation treated as polarization," *Biological Cybernetics*, vol. 97, pp. 341–349, 12 2007.

[70] L. S. Smith and D. S. Fraser, "Sound feature detection using leaky integrate-and-fire neurons."

[71] R. A. Vazquez and A. Cachón, *Integrate and Fire Neurons and their Application in Pattern Recognition*. [IEEE], 2010.

[72] S. A. Chaturvedi, S. Chaturvedi, M. Boudjelal, and A. A. Khurshid, "Image segmentation using leaky integrate and fire model of spiking neural network," 2012. [Online]. Available: https://www.researchgate.net/publication/316284219

[73] E. Doutsi, L. Fillatre, M. Antonini, and P. Tsakalides, "Dynamic image quantization using leaky integrate-and-fire neurons," *IEEE Transactions on Image Processing*, vol. 30, pp. 4305–4315, 4 2021.

[74] P. Mullowney and S. Iyengar, "Parameter estimation for a leaky integrate-and-fire neuronal model from isi data," *Journal of Computational Neuroscience*, vol. 24, pp. 179–194, 4 2008.

[75] J. Liu, Y. Huang, Y. Luo, J. Harkin, and L. McDaid, "Bio-inspired fault detection circuits based on synapse and spiking neuron models," *Neurocomputing*, vol. 331, pp. 473–482, 2 2019.

[76] M. Chu, B. Kim, S. Park, H. Hwang, M. Jeon, B. H. Lee, and B. G. Lee, "Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron," *IEEE Transactions on Industrial Electronics*, vol. 62, pp. 2410–2419, 2015.

[77] J. Q. Yang, R. Wang, Z. P. Wang, Q. Y. Ma, J. Y. Mao, Y. Ren, X. Yang, Y. Zhou, and S. T. Han, "Leaky integrate-and-fire neurons based on perovskite memristor for spiking neural networks," *Nano Energy*, vol. 74, 8 2020.

[78] M. A. Nahmias, B. J. Shastri, A. N. Tait, and P. R. Prucnal, "A leaky integrate-and-fire laser neuron for ultrafast cognitive computing," *IEEE Journal on Selected Topics in Quantum Electronics*, vol. 19, 2013.

[79] D. Chatterjee and A. Kottantharayil, "A cmos compatible bulk finfet-based ultra low energy leaky integrate and fire neuron for spiking neural networks," *IEEE Electron Device Letters*, vol. 40, pp. 1301–1304, 8 2019.

[80] M. J. Rozenberg, O. Schneegans, and P. Stoliar, "An ultra-compact leaky-integrate-and-fire model for building spiking neural networks," *Scientific reports*, vol. 9, p. 11123, 7 2019.

[81] D. I. Standage and T. P. Trappenberg, "Differences in the subthreshold dynamics of leaky integrate-and-fire and hodgkin-huxley neuron mo els."

[82] M. S. Chaturvedi and A. A. Khurshid, "Comparison of support vector machine and leaky-integrated fire snn model for object recognition." [Online]. Available: https://www.researchgate.net/publication/316283981

[83] E. M. Izhikevich, "Simple model of spiking neurons," pp. 1569–1572, 11 2003.

[84] E. M. Izhikevich., "Which model to use for cortical spiking neurons?" *IEEE Transactions on Neural Networks*, vol. 15, pp. 1063–1070, 9 2004.

[85] S. Haghiri, A. Zahedi, A. Naderi, and A. Ahmadi, "Multiplierless implementation of noisy izhikevich neuron with low-cost digital design," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, pp. 1422–1430, 12 2018.

[86] A. Elnabawy, H. Abdelmohsen, M. Moustafa, M. Elbediwy, A. Helmy, and H. Mostafa, *A Low Power CORDIC-Based Hardware Implementation of Izhikevich Neuron Model*.

[87] K. L. Rice, M. A. Bhuiyan, T. M. Taha, C. N. Vutsinas, and M. C. Smith, "Fpga implementation of izhikevich spiking neural networks for character recognition," 2009, pp. 451–456.

[88] R. Antonio, V. Universidad, L. S. México, and R. A. Vázquez, "Izhikevich neuron model and its application in pattern recognition spiking neurons and their applications to pattern recognition view project izhikevich neuron model and its application in pattern recognition," 2010. [Online]. Available: https://www.researchgate.net/publication/285718078

[89] A. Luna-Álvarez, D. Mújica-Vargas, and M. Mejía-Lavalle, "Convolutional model with classification through izhikevich neuron," *Research in Computing Science*, vol. 148, pp. 65–80, 12 2019.

[90] M. J. Skocik and L. N. Long, "On the capabilities and computational costs of neuron models," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, pp. 1474–1483, 2014.

[91] S. Valadez-Godínez, H. Sossa, and R. Santiago-Montero, "How the accuracy and computational cost of spiking neuron simulation are affected by the time span and firing rate," *Computacion y Sistemas*, vol. 21, pp. 841–861, 2017.

[92] S. Valadez-Godinez, H. Sossa, and R. Santiago-Montero, "On the accuracy and computational cost of spiking neuron implementation," *Neural Networks*, vol. 122, pp. 196–217, 2 2020.

[93] W. Gerstner, R. Ritz, and J. L. V. Hemmen, "Why spikes? hebbian learning and retrieval of time-resolved excitation patterns," pp. 503–515, 1993.

[94] R. Jolivet, T. J. Lewis, and W. Gerstner, "The spike response model: A framework to predict neuronal spike trains." [Online]. Available: http://diwww.epfl.ch/mantrahttp://www.cns.nyu.edu/˜tlewis

[95] S. M. Bohte, "Efficient spike-coding with multiplicative adaptation in a spike response model," 2012.

[96] A. Ourdighi and A. Benyettou, "An efficient spiking neural network approach based on spike response model for breast cancer diagnostic," 2016.

[97] L. Gammaitoni, P. Jung, and F. Marchesoni, "Stochastic resonance," 1998.

[98] P. H‰nggi, "Stochastic resonance in biology how noise can enhance detection of weak signals and help improve biological information processing," 2002.

[99] R. F. Fox, "Stochastic versions of the hodgkin-huxley equations," *Biophysical Journal*, vol. 72, pp. 2068–2074, 1997.

[100] N. Kasabov, "To spike or not to spike: A probabilistic spiking neuron model," *Neural Networks*, vol. 23, pp. 16–19, 1 2010.

[101] N. Kasabov, K. Dhoble, N. Nuntalid, and A. Mohemmed, "Evolving probabilistic spiking neural networks for spatio-temporal pattern recognition: A preliminary study on moving object recognition." [Online]. Available: http://www.kedri.info

[102] O. Y. Sinyavskiy and A. I. Kobrin, "Generalized stochastic spiking neuron model and extended spike response model in spatial-temporal pulse pattern detection task," *Optical Memory and Neural Networks (Information Optics)*, vol. 19, pp. 300–309, 2010.

[103] T. Wu, S. Fu, L. Cheng, R. Zheng, X. Wang, X. Kuai, and G. Yang, "A simple probabilistic spiking neuron model with hebbian learning rules," 2012.

[104] P. Lansky and J. P. Rospars, "Ornstein-uhlenbeck model neuron revisited," pp. 397–406, 1995.

[105] S. Ditlevsen and P. Lansky, "Estimation of the input parameters in the feller neuronal model," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 73, 2006.

[106] P. Lansky and S. Ditlevsen, "A review of the methods for signal estimation in stochastic diffusion leaky integrate-and-fire neuronal models," pp. 253–262, 11 2008.

[107] A. Galves and E. Löcherbach, "Infinite systems of interacting chains with memory of variable length-a stochastic model for biological neural nets," *Journal of Statistical Physics*, vol. 151, pp. 896–921, 6 2013.

[108] H. C. Tuckwell and R. Rodriguez, "Analytical and simulation results for stochastic fitzhugh-nagumo neurons and neural networks," pp. 91–113, 1998.

[109] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, pp. 5872–5878, 7 2013.

[110] M. Al-Shedivat, R. Naous, G. Cauwenberghs, and K. N. Salama, "Memristors empower spiking neurons with stochasticity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, pp. 242–253, 6 2015.

[111] J. L. RossellÓ, V. Canals, A. Morro, and A. Oliver, "Hardware implementation of stochastic spiking neural networks," *International Journal of Neural Systems*, vol. 22, 8 2012.

[112] G. A. F. Guerra and S. B. Furber, "Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems," *Frontiers in Neuroscience*, vol. 11, 12 2017.

[113] S. Schliebs, A. Mohemmed, and N. Kasabov, *Are Probabilistic Spiking Neural Networks Suitable for Reservoir Computing?* IEEE.

[114] N. Nuntalid, K. Dhoble, and N. Kasabov, "Eeg classification with bsa spike encoding algorithm and evolving probabilistic spiking neural network," vol. 7062 LNCS, 2011, pp. 451–460.

[115] J. Huxter, N. Burgess, and J. O'Keefe, "Independent rate and temporal coding in hippocampal pyramidal cells," *Nature*, vol. 425, pp. 824–828, 10 2003.

[116] B. J. Richmond and L. M. Optican, "Temporal encoding of two-dimensional patterns by single units in primate inferior temporal cortex. ii. quantification of response waveform," 1987. [Online]. Available: www.physiology.org/journal/jn

[117] Mainen and Sejnowski, "Reliability of spike timing in neocortical neurons." [Online]. Available: www.sciencemag.org

[118] B. Rueckauer and S.-C. Liu, *Conversion of analog to spiking neural networks using sparse temporal coding*. IEEE, 2018.

[119] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 3227–3235, 7 2018.

[120] S. M. Bohte, J. N. Kok, and H. L. Poutrã, "Error-backpropagation in temporally encoded networks of spiking neurons," pp. 17–37, 2002. [Online]. Available: www.elsevier.com/locate/neucom

[121] S. A. Lobov, A. V. Chernyshov, N. P. Krilova, M. O. Shamshin, and V. B. Kazantsev, "Competitive learning in a spiking neural network: Towards an intelligent pattern classifier," *Sensors (Switzerland)*, vol. 20, 1 2020.

[122] S. Yin, S. K. Venkataramanaiah, G. K. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, and J.-S. Seo, "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations."

[123] M. Kiselev, *Rate Coding vs. Temporal Coding – Is Optimum Between?*

[124] D. Heeger, "Poisson model of spike generation," 2000.

[125] S. Thorpe and J. Gautrais, "Rank order coding," pp. 113–118, 1998.

[126] A. Delorme, L. Perrinet, and S. J. Thorpe, "Networks of integrate-and-fire neurons using rank order coding b: Spike timing dependent plasticity and emergence of orientation selectivity," p. 545, 2001.

[127] S. Loiselle, J. Rouat, D. Pressnitzer, and S. Thorpe, "Exploration of rank order coding with spiking neural networks for speech recognition." Institute of Electrical and Electronics Engineers (IEEE), 1 2006, pp. 2076–2080.

[128] N. K. Kasabov, *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*. [Online]. Available: http://www.springer.com/series/15821

[129] R. S. Petersen, S. Panzeri, and M. E. Diamond, "Population coding of stimulus location in rat somatosensory cortex millisecond scale can be a mechanism for information transmission about dynamic stimulus features both in nonmammalian neural structures (bialek et al," pp. 503–514, 2001.

[130] A. Onken, P. P. C. R. Karunasekara, C. Kayser, and S. Panzeri, "Understanding neural population coding: Information theoretic insights from the auditory system," *Advances in Neuroscience*, vol. 2014, pp. 1–14, 10 2014.

[131] I. Dean, N. S. Harper, and D. McAlpine, "Neural population coding of sound level adapts to stimulus statistics," *Nature Neuroscience*, vol. 8, pp. 1684–1689, 12 2005.

[132] S. Soltic and N. Kasabov, "Knowledge extraction from evolving spiking neural networks with rank order population coding," *International Journal of Neural Systems*, vol. 20, pp. 437–445, 12 2010.

[133] Z. Pan, J. Wu, Y. Chua, M. Zhang, and H. Li, "Neural population coding for effective temporal classification," 9 2019. [Online]. Available: http://arxiv.org/abs/1909.08018

[134] H. R. Schindler, "Delta modulation better communications are possible when the signals produced by the human voice are changed into a series of binary digits. one conversion method offers special advantages."

[135] B. Petro, N. Kasabov, and R. M. Kiss, "Selection and optimization of temporal spike encoding methods for spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, pp. 358–370, 2 2020.

[136] F. Gers, N. E. Nawa, M. Hough, H. D. Garis, and M. Korkin, "Spiker: Analog waveform to digital spiketrain conversion in atrapos;s artificial brain (cam-brain) project digital microbiology lab view project codi-a cellular automata based neural net model view project michael korkin independent researcher," 1999. [Online]. Available: http://www.stanford.edu/mhough http://www.hip.atr.co.jp/fdegaris,xnawag http://www.genobyte.com http://www.idsia.ch/felix

[137] B. Schrauwen and J. V. Campenhout, "Bsa, a fast and accurate spike train encoding scheme," vol. 4, 2003, pp. 2825–2830.

[138] J. Dupeyroux, "A toolbox for neuromorphic sensing in robotics," 3 2021. [Online]. Available: http://arxiv.org/abs/2103.02751

[139] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, "Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems," *Frontiers in Neuroscience*, vol. 15, 3 2021.

[140] C. D. Schuman, J. S. Plank, G. Bruer, and J. Anantharaj, "Non-traditional input encoding schemes for spiking neuromorphic systems." [Online]. Available: http://energy.gov/downloads/doe-public-

[141] T. Natschläger and B. Ruf, "Spatial and temporal pattern analysis via spiking neurons," *Network: Computation in Neural Systems*, vol. 9, 1 1998.

[142] J. Sjöström and W. Gerstner, "Spike-timing dependent plasticity," *Scholarpedia*, vol. 5, 2010.

[143] T. Masquelier, R. Guyonneau, and S. J. Thorpe, "Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains," *PLoS ONE*, vol. 3, 1 2008.

[144] J.-M. Fellous, "Discovering spike patterns in neuronal responses," *Journal of Neuroscience*, vol. 24, 3 2004.

[145] M. C. W. van Rossum, G. Q. Bi, and G. G. Turrigiano, "Stable hebbian learning from spike timing-dependent plasticity," *The Journal of Neuroscience*, vol. 20, 12 2000.

[146] R. Guyonneau, R. VanRullen, and S. J. Thorpe, "Neurons tune to the earliest spikes through stdp," *Neural Computation*, vol. 17, 4 2005.

[147] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Computational Biology*, vol. 3, 2 2007.

[148] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex," *Neural Computation*, vol. 13, pp. 1255–1283, 6 2001.

[149] E. Ahissar and A. Arieli, "Seeing via miniature eye movements: A dynamic hypothesis for vision," *Frontiers in Computational Neuroscience*, 10 2012.

[150] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," 1 2019. [Online]. Available: http://arxiv.org/abs/1901.09948

[151] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, 6 1989.

[152] M. C. W. van Rossum, "A novel spike distance," *Neural Computation*, vol. 13, 4 2001.

[153] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral Cortex*, vol. 17, 10 2007.

[154] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in Neural Circuits*, vol. 9, 1 2016.

[155] H. T. Ito, "Frequency-dependent signal transmission and modulation by neuromodulators," *frontiers in Neuroscience*, vol. 2, 12 2008.

[156] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, 5 2020.

[157] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in Neuroscience*, vol. 11, 6 2017.

[158] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, 11 2002.

[159] A. M. George, D. Banerjee, S. Dey, A. Mukherjee, and P. Balamurali, "A reservoir-based convolutional spiking neural network for gesture recognition from dvs input." IEEE, 7 2020.

[160] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/82f2b308c3b01637c607ce05f52a2fed-Paper.pdf

[161] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers in Neuroscience*, vol. 12, 8 2018.

[162] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Communications*, vol. 11, 12 2020.

[163] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks."

[164] A. Brasoveanu, M. Moodie, and R. Agrawal, "Neko: a library for exploring neuromorphic learning rules," vol. 2657. CEUR-WS, 2020, pp. 1–9.

[165] M. Traub, M. V. Butz, R. H. Baayen, and S. Otte, "Learning precise spike timings with eligibility traces," 5 2020. [Online]. Available: http://arxiv.org/abs/2006.09988

[166] W. V. D. Veen.

[167] F. Scherr, C. Stöckl, and W. Maass, "One-shot learning with spiking neural networks," 2020.

[168] M. Litzenberger, B. Kohn, A. Belbachir, N. Donath, G. Gritsch, H. Garn, C. Posch, and S. Schraml, "Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor." IEEE, 2006.

[169] K. Boahen, "A burst-mode word-serial address-event link—i: Transmitter design," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, 7 2004.

[170] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-Based Neuromorphic Systems*. John Wiley Sons, Ltd, 1 2015.

[171] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 db 15 s latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, 2008.

[172] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proceedings of the IEEE*, vol. 102, 10 2014.

[173] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, 8 2015.

[174] T. Delbruck, B. Linares-Barranco, E. Culurciello, and C. Posch, "Activity-driven, event-based vision sensors." IEEE, 5 2010.

[175] E. Fossum, "Cmos image sensors: electronic camera-on-a-chip," *IEEE Transactions on Electron Devices*, vol. 44, 1997.

[176] Y. Nozaki and T. Delbruck, "Temperature and parasitic photocurrent effects in dynamic vision sensors," *IEEE Transactions on Electron Devices*, vol. 64, 8 2017.

[177] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[178] G. Gallego, M. Gehrig, and D. Scaramuzza, "Focus is all you need: Loss functions for event-based vision," 4 2019. [Online]. Available: http://arxiv.org/abs/1904.07235

[179] M. A. R. Ahad, J. K. Tan, H. Kim, and S. Ishikawa, "Motion history image: its variants and applications," *Machine Vision and Applications*, vol. 23, 3 2012.

[180] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "Hots: A hierarchy of event-based time-surfaces for pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, 7 2017.

[181] H. Rebecq, R. Ren´, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera multimedia material," p. 1, 2019. [Online]. Available: http://rpg.ifi.uzh.ch/e2vid.

[182] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-dvs and mnist-dvs. their history, how they were made, and other details," *Frontiers in Neuroscience*, vol. 9, 2015.

[183] A. Amir, B. Taba, D. Berg, T. Melano, J. Mckinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system."

[184] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, 5 2018.

[185] C. Pehle and J. E. Pedersen, "Norse - A deep learning library for spiking neural networks," Jan. 2021, documentation: https://norse.ai/docs/. [Online]. Available: https://doi.org/10.5281/zenodo.4422025

[186] P. Vincent-Lamarre, M. Calderini, and J.-P. Thivierge, "Learning long temporal sequences in spiking networks by multiplexing neural oscillations," *Frontiers in Computational Neuroscience*, vol. 14, 9 2020.

[187] T. Masquelier, E. Hugues, G. Deco, and S. J. Thorpe, "Oscillations, phase-of-firing coding, and spike timing-dependent plasticity: An efficient learning scheme," *Journal of Neuroscience*, vol. 29, 10 2009.