Active dendrites for continual learning in a time-to-first-spike spiking neural network architecture

Mitigating the catastrophic forgetting problem in time-to-first-spike networks using bio-inspired principles

by

Lorenzo Pes

Student Name	Student Number
Lorenzo Pes	5604958

Supervisor: K. Makinwa
Daily Supervisor: C. Frenkel
Faculty: Faculty of Electrical Engineering, Delft

TU Delft

Abstract

Spiking neural networks (SNNs) are a new generation of neural networks aiming at reducing the power consumption of conventional artificial intelligence systems by mimicking the behaviour of biological neurons found in the human brain. To achieve this goal, SNNs mimic the propagation of information observed in biological neurons through the use of discrete events known as spikes. Historically, different theories have been proposed to explain how information is encoded into these spike events. One such theory is time-to-first-spike (TTFS) coding, which offers valuable opportunities for low-power and low-latency hardware implementations.

Nonetheless, networks of spiking neurons still miss a characteristic of learning observed in human beings. Specifically, they are unable to learn different tasks in a sequential fashion without incurring in the problem of *catastrophic forgetting*. Indeed, while these networks achieve state-of-the-art results in a vast number of problems, they require a full retraining of the network as new tasks need to be learned. This problem not only highlight a difference from biological systems, but also limits the applicability of such systems in environments which require adaptation to new tasks.

Currently, there exists no hardware that is capable of mitigating the problem of *catastrophic forgetting* while leveraging the low-power and low-latency opportunities offered by TTFS coding. To overcome this research gap, we conducted a literature review of proposed solutions to the problem of catastrophic forgetting in both the artificial and spiking neural network domains. The aim of this review it to uncover biologically inspired solutions to the problem of catastrophic forgetting which could be applied to TTFS-encoded spiking networks. Furthermore, to implement a digital hardware accelerator capable of incorporating the requirements of the selected solution, we summarized key architectures for event-based SNNs.

This thesis presents a novel neural model based on a spike response model (SRM) with a Rel-PSP kernel, which is enhanced with *active dendrites*. The proposed solution successfully mitigates the problem of catastrophic forgetting in a typical continual learning setup, in which the network is trained over different tasks in a sequential fashion, i.e. one task after the other. Additionally, we designed a digital hardware architecture that implements the proposed solution in a Xilinx Zynq-7020 SoC FPGA.

Our solution is capable of learning the first five digits of the N-MNIST dataset in a sequential fashion, resulting in a final average accuracy of 100% across all tasks. Conversely, the same model without active dendrites achieves an accuracy of only 23%, which is close to random guessing, thereby demonstrating a successful mitigation of catastrophic forgetting with the proposed solution. Additionally, our digital hardware implementation is capable of classifying a sample image of the dataset in an average time of $117\ \mu s$ while consuming $232\ mW$ at a clock frequency of $125\ MHz$. The proposed architecture uses 74% of the LUTs, 28% of the FFs, and 32% of the BRAM available in the FPGA.

Acknowledgment

First and foremost, I would like to express my profound gratitude to my thesis advisor Charlotte Frankel, who granted me the freedom to choose this research topic and provided continuous support from both a theoretical and technical perspective. Her insights throughout the entire journey were incredibly valuable and played a fundamental role in making this work possible.

I extend my gratitude to Nicolas Chauvaux for providing great insights about the digital hardware implementation of this thesis and for enduring my complaints during challenging and stressful times in our shared office.

Furthermore, I wish to express my gratitude to Malu Zhang for providing the baseline code of his TTFS-encoded neuron model. This code served as a valuable starting point for the software development of the solution proposed in this work.

Additionally, I am deeply grateful for the friendships I have cultivated throughout my academic journey at Delft University of Technology. In particular, I am very thankful to my microelectronics colleagues Andrea, Giorgos, and Jacopo, who made this thesis, as well as the whole experience at TUDelft, an unforgettable period of my life. Also, I would like to extend my gratitude to all my other friends in Delft, including Alessia, Margherita and Riccardo for providing unforgettable dinners.

I also want to express my profound gratitude to my girlfriend Elena, for deeply understanding the sacrifices I have made to make this work possible while maintaining a long-distance relationship.

This thesis is dedicated to both my parents, who have supported and encouraged the pursuit of my passions and interests since I was very young. I would also like to express my gratitude to them for giving me the opportunity to leave my native country at the age of nineteen. The opportunity to travel the world alone has played a significant role in shaping the person I have become today.

Contents

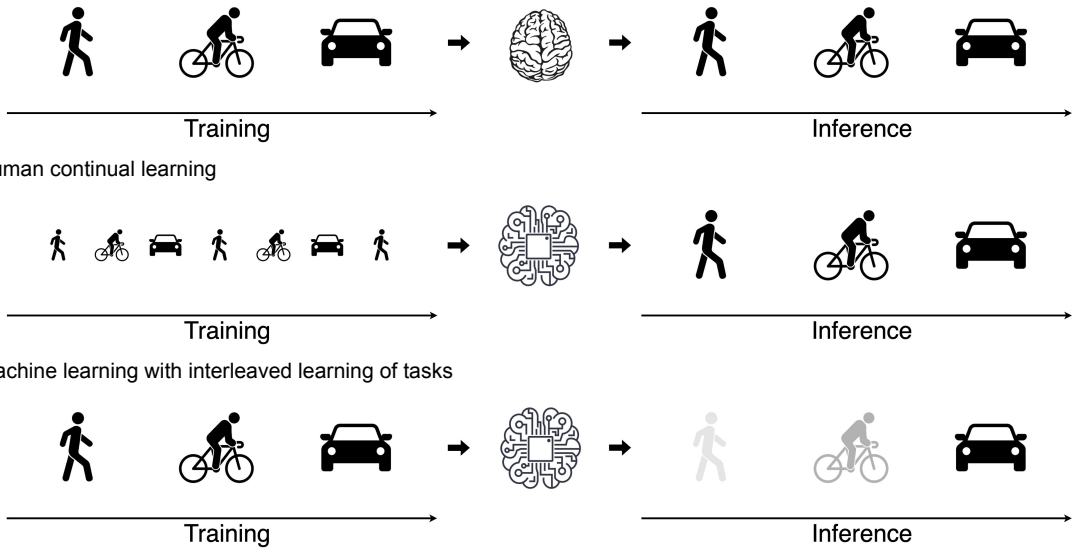
List of Figures

List of Tables

1

Introduction

As human beings experience the physical world, they have the inherent ability to learn different tasks. Take, for instance, consecutive learning experiences, from the initial steps of walking to the ability of driving a bike without falling, and finally to the more complex task of driving a car. Although the act of driving a bike shares some similarities to walking, e.g. both tasks require similar movements of the legs, humans have the innate ability to learn a new task while incorporating information from previously learned tasks, without forgetting how to perform them. This ability, called *continual learning (CL)* in the deep learning community, is visually depicted in Fig. 1.1a.



(a) Human continual learning

(b) Machine learning with interleaved learning of tasks

(c) Machine learning with sequential learning of tasks (*catastrophic forgetting*)

Figure 1.1: Human vs machine learning. **(a)** The human brain can learn different tasks in sequential order without forgetting them. **(b)** Machine learning with *stochastic gradient descent (SGD)*: training examples are provided in an interleaved fashion and tasks are learned simultaneously. **(c)** Machine learning in a sequential training environment results in forgetting initial tasks, e.g. walking and biking.

Inspired by the capabilities of the human brain, artificial neural networks (ANNs) have demonstrated astonishing capabilities in recognizing patterns [1], detecting objects from images and videos [2, 3], processing natural language [4, 5], and recogniz-

ing speech [6]. These remarkable achievements were made possible by the introduction of *back-propagation (BP)* [7] and *stochastic gradient descent (SGD)* [8], which require to present each task to the network in an interleaved fashion as depicted in Fig. 1.1b. Indeed, following our initial metaphor, whereas the human brain can sequentially learn to walk, ride a bike, and drive a car, current training methodologies using SGD require to present examples of each task in an interleaved manner. It is as if one were to learn a little about walking, then a little about driving a car, and then a little about riding a bike, repeating this process iteratively until all tasks are learned. Following this approach, incorporating a new task into the neural network requires aggregating the new data with the data from previous tasks, followed by a retraining of the network from scratch. The first trace of this limitation appeared in 1989 through the work of McCloskey [9]. In this work, the term *catastrophic inference* (or *catastrophic forgetting*) was introduced to indicate the inability of deep neural networks to learn new tasks sequentially. An example of this problem is depicted in Fig. 1.1c.

Moreover, beyond the limitation imposed by catastrophic forgetting, the power consumption of current *Artificial Intelligent (AI)* machines is approximately two orders of magnitude higher than that of the brain when solving the same task [10]. With the aim of reducing the power consumption of AI systems, researchers started to develop a new generation of neural networks called spiking neural networks (SNNs), which attempt to mimic the dynamics of biological neurons. This novel approach, known as *neuromorphic computing*, seeks to emulate the remarkable efficiency and adaptability of the brain in information processing to reduce the power consumption of AI hardware. The fundamental difference between SNNs and ANNs lies in the way information is represented and processed. For example, in traditional ANNs, the information is represented using real-valued numbers. On the other hand, SNNs use discrete events, called spikes, to propagate information throughout the network. This approach is inspired by the action potential generated by neural cells to communicate information to other neurons. The difference between these two approaches is shown in Fig. 1.2 for a single neuron.

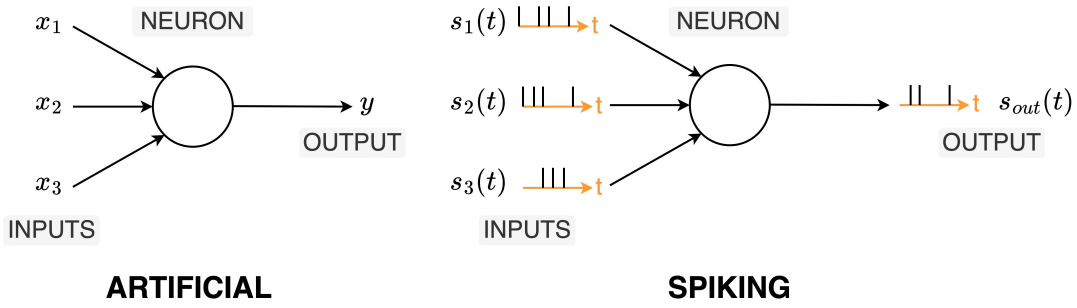


Figure 1.2: Artificial vs spiking neuron. On the left side, x_1, x_2 and x_3 are the real-valued inputs of an artificial neuron. The symbol y is the real-valued output. On the right side, $s_1(t)$, $s_2(t)$ and $s_3(t)$ are the input spike trains of a spiking neuron. The symbol $s_{out}(t)$ is the output spike train.

How information is efficiently encoded into biological spikes has been extensively researched in the field of computational neuroscience, and it is still an open question. Historically, different coding schemes have been proposed to explain the mechanism of information transmission in neural cells. One such scheme is rate coding, which

is the dominant model in neuroscience and SNNs. It uses spiking rates to represent information and has been experimentally observed in sensory systems, such as the visual and motor cortices [11]. Despite its simplicity and robustness, rate coding requires an observation window to calculate the mean rate, resulting in slow information transmission and long processing times. On the other hand, another scheme, referred to as time-to-first-spike (TTFS) coding, has been hypothesized to account for areas of the brain that require a fast and efficient response. Numerous experiments highlighted the plausibility of this encoding mechanism in nervous systems such as the retina and tactile afferents [12]. This coding scheme encodes information into the spike time from an initial observation reference, and each neuron spikes at most once. Compared to the aforementioned rate coding, the behavior of TTFS coding is characterized by a low spiking activity (i.e. high sparsity), which holds significant potential for low power consumption, particularly when implemented into custom event-based hardware. Moreover, because the information is encoded in the time of the first spike, network inference can potentially be made extremely fast by eliminating the necessity to wait for all neurons to spike.

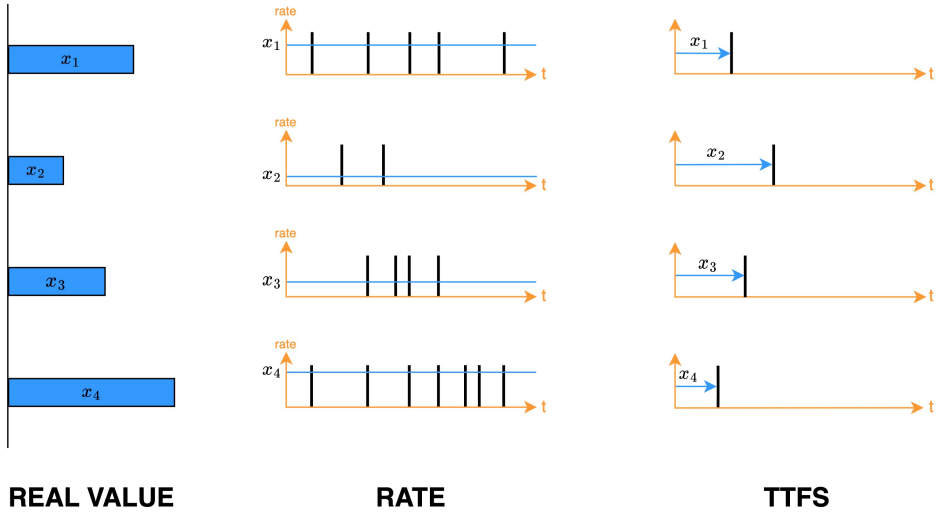


Figure 1.3: Encoding mechanisms. A real value can be converted into rate or TTFS coding. In the former, the value is encoded in the firing rate. In the latter, the value is encoded in the time of the spike from an initial reference: the lower the time until the first spike, the higher the value.

Nowadays, there is a strong need for fast and power-efficient CL hardware that can incorporate new tasks without a complete retraining of the network. However, despite the potential of TTFS coding for efficient hardware, there exists no prior proposal for learning algorithms aimed at mitigating the problem of catastrophic forgetting in TTFS encoded neural networks. This deficiency extends to the domain of hardware accelerators that could be effectively used in continual learning situations that require low power and fast inference. To fill this gap, the main research question of this thesis can be formulated as follows:

Is it feasible to formulate a learning algorithm capable of mitigating the problem of catastrophic forgetting in TTFS encoded networks and design its corresponding hardware accelerator following a neuromorphic approach ?

To address this question, a literature review has been conducted to investigate solutions aimed at mitigating catastrophic forgetting in both the realms of ANNs and SNNs. This exploration seeks to uncover strategies that are not only rooted in traditional computational methods, but also inspired by the complex biology of neural systems. Furthermore, to address the design of a corresponding hardware accelerator, a review focusing on existing event-based architectures for continual learning and time-to-first-spike coding has been conducted. This review aims to design an accelerator that can incorporate the solution to the catastrophic forgetting problem while accommodating the requirements of TTFS coding, resulting in the following sub-problems of the main research question:

1. *What specific biologically-inspired computational mechanisms could be leveraged to solve the problem of catastrophic forgetting ?*
2. *Does TTFS coding offer an effective advantage in terms of power consumption and latency compared to other coding schemes?*
3. *Can the inherent characteristics of TTFS encoded neural models be used to develop a learning algorithm that effectively mitigates catastrophic forgetting while easing hardware requirements ?*
4. *How do different architectural choices of the neural model impact hardware requirements ?*

The main contribution of this thesis is two-fold. First, it proposes a novel neural model inspired by the biological concept of *active dendrites* [13], which can effectively mitigate the problem of catastrophic forgetting in time-to-first-spike neural networks. Second, it provides an efficient hardware accelerator design that can effectively accommodate for the unique characteristics of the proposed model. The efficacy of the proposed solution to retain previously learned tasks has been evaluated with a sequential classification task based on the neuromorphic MNIST (N-MNIST) digit dataset [14]. The proposed solution successfully learns the first five digits in sequential order, achieving an average accuracy of 100%. On the contrary, when the same model is used without the suggested approach, the average accuracy drops significantly to 23%, thus demonstrating the effective mitigation of the catastrophic forgetting problem. Furthermore, the hardware accelerator enhanced with active dendrites can classify an image in an estimated average time of $117\ \mu s$ at $f_{clk} = 125\ MHz$ while consuming $231\ mW$ of power on a Zynq-7020 SoC FPGA, and using 72 % of the available resources. To guide the reader through a comprehensive exploration of the proposed solution, this thesis is organized into several chapters.

Chapter 2 introduces the reader to the foundational background material required to comprehend the proposed solution. Initially, an exploration of the biological anatomy and operating principles of biological neurons is presented. This material will serve as a basis to understand the different encoding mechanisms used by biological neurons to transmit information, along with an exploration of the hardware and software implications associated with each mechanism. Following this introduction, the chapter explains the most popular neuron models used by the neuromorphic community. Specifically, it will provide mathematical formulations for the leaky integrate-and-fire

(LIF) model and the spike response model (SRM). Furthermore, the chapter elaborates on the conventional layered architecture and the algorithm used to train such a structure. Moving forward, a literature review of existing software approaches addressing the problem of catastrophic forgetting in both SNNs and ANNs is provided with the aim of uncovering a biologically plausible mechanism to mitigate the problem of catastrophic forgetting in a TTFS network. Finally, this chapter offers an overview of general hardware architectures for TTFS networks and continual learning.

Chapter 3 presents the design of the proposed solution from both the software and the hardware perspectives. It begins by laying out the mathematical basis of the proposed neuron model, followed by an explanation of the network architecture and the training process. Furthermore, this chapter provides a detailed explanation of the hardware architecture and its subcomponents, including controllers, the neural model, and the neural cluster.

Chapter 4 discusses the results obtained from both the software simulation and the hardware implementation. It starts by introducing the case study used to gauge the efficacy of the proposed model. Afterwards, it provides a discussion of the software results in terms of the ability of the proposed model to retain previously learned tasks. Furthermore, the hardware results of the implemented solution will be discussed in terms of latency, resource utilization, power consumption, and energy efficiency.

Chapter 5 presents the conclusion of this thesis and explores potential avenues for future work with the proposed model.

2

Background material

This chapter presents the fundamental knowledge required to comprehend the proposed solution to the main research question of this thesis. In Section 2.1, we will introduce the anatomy and operating principles of biological neurons. This foundational knowledge will serve as the basis for Section 2.2, where we examine the different encoding schemes employed by neurons to transmit information and their implications in the context of machine learning and hardware implementations. In Section 2.3, we will introduce the most popular models employed by the neuromorphic community to mimic the behavior of biological neurons. In Section 2.4, we will explore both the conventional architectures used to implement networks of spiking neurons and the training algorithm employed to train such structures. This knowledge will provide the foundation to understand the problem of catastrophic forgetting discussed in Section 2.5. In addition to a formal definition of the problem, in this section, we will provide information on evaluation datasets and discuss the existing literature on the problem of catastrophic forgetting. Within this discussion, we will highlight the significant role that active dendrites play in enabling spiking networks to learn continually. Finally, in Section 2.6, we will introduce the reader to the general hardware architectures for continual learning and TTFS-encoded networks.

2.1. Principles and anatomy of biological neurons

The cerebral cortex, often referred to as the gray matter of the brain, is a complex and intricate structure of computational units known as neurons, among which pyramidal neurons are the most abundant [15]. As illustrated in Fig. 2.1, pyramidal neurons consist of the following components:

- **Synapse:** connection between the axon terminal of one neuron and the dendrites of another neuron.
- **Distal dendrites:** also called active dendrites, are tree-shaped structures containing thousands of synapses that receive signals from neurons in the distal regions of the brain. They perform non-linear local processing of contextual inputs [16]. There are two classes of distal dendrites: **basal**, extending from the base of the neuron, and **apical**, extending from the apex.

- **Proximal dendrites:** tree-shaped structures, located close to the neuron's soma, contain thousands of synapses that receive signals from nearby neurons and perform a linear processing of incoming signals.
- **Soma:** the main body of the neuron, which is responsible for processing post-synaptic signals from proximal and basal dendritic trees.
- **Axon:** a nerve fiber that transmits the processed signals from the soma to neighboring neurons through a series of electrical pulses known as spikes.

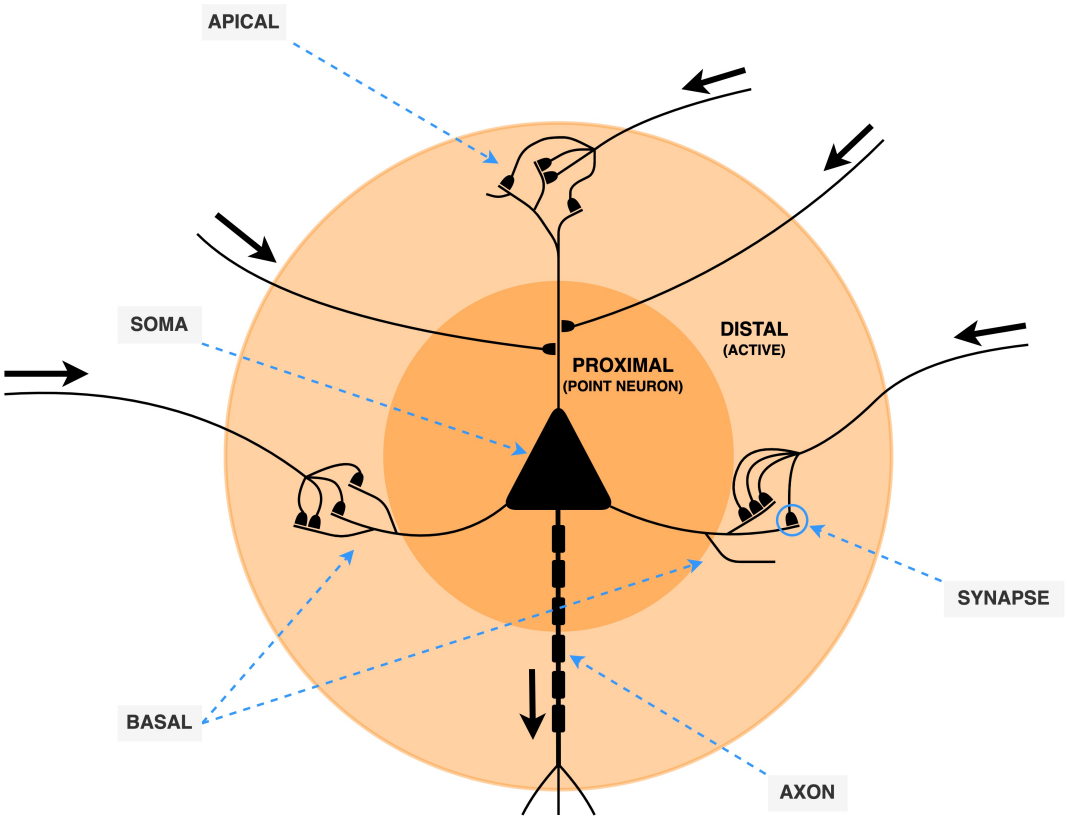


Figure 2.1: Anatomy of pyramidal neurons. A pyramidal neuron consists of two different types of dendritic branches: distal, also known as active dendrites, and proximal. The former contains thousands of connections from neurons located in distal regions of the brain, whereas the latter contains connections from neighboring neurons. These connections are called synapses and link the axon of one neuron to the dendrites of another. The triangle in the center of the image represents the neuron's soma. The branches extending from the base of the triangle are called basal dendrites, whereas the ones extending from the apex of the triangle are called apical dendrites. The black arrows represent the flow of information in the neuron. As one can see, the dendrites receive signals from other neurons while the axon sends the processed signal to other neurons.

The soma of a neuron exhibits a difference in the electric potential between its inner part and the outer environment, also known as the *membrane voltage*. This difference arises from the motion of ions like sodium (Na+), potassium (K+), and chloride (Cl-) flowing in and out of the cell. When at rest, i.e. balanced ions flow, the potential sits at -70 mV, also known as the *resting voltage*. Whenever the neuron receives a stimulus from another neuron through a synaptic connection, the membrane voltage

leaves its resting state[1]. If the post-synaptic signal is strong enough, the membrane can reach a specific voltage called *threshold voltage* after which the neuron emits an *action potential* as shown in Fig. 2.2. The action potential is an abrupt increase in the electrical potential of the neural cell characterized by a depolarization and a repolarization phase, after which the neuron returns to its resting potential. This action potential travels through the axon fiber until it reaches the synapses located in the dendritic trees of other neurons.

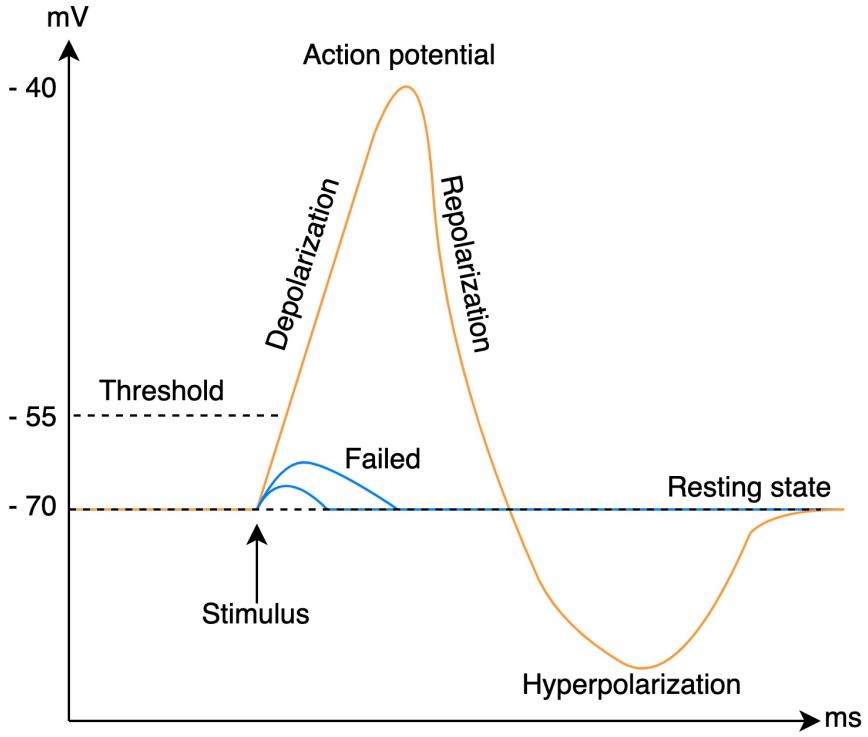


Figure 2.2: Action potential. Following a post-synaptic stimulus, the membrane potential, represented by the orange line, reaches the threshold voltage. After this event, the neuron cell *depolarizes* until it reaches a peak potential and *repolarizes* until it reaches the *hyperpolarization* phase. During this stage, the neuron is insensitive to incoming stimuli. Following this phase, the potential goes back to the resting state and becomes sensitive to incoming stimuli again. If the stimulus is not strong enough, the action potential is not elicited, resulting in a failed interaction, represented by the blue lines.

It is a well-known fact that synaptic connections between neurons play an essential role in retaining the information obtained during learning, as they can modulate how information is transferred to the membrane potential [17]. Specifically, a synapse can undergo a change in strength, which either enhances or weakens the signal transmitted from the pre-synaptic neuron. This idea, referred to as *synaptic plasticity*, constitutes the core concept of learning in networks of neurons.

[1]The input signal has to be strong enough to move the membrane potential from its resting state and the neuron must not be in its hyperpolarization phase.

2.2. Neural coding

Long before the advent of neural networks as a computing paradigm, humans have wondered about the language of neurons. The concept of neural coding, or the idea that patterns of electrical activity within the brain might represent the alphabet of neural cells, has deep historical roots. This idea was first proposed by Cajal in 1909 in his work [18]. Although his intuitions were groundbreaking, a formal mathematical model to describe the neural language of neurons was still missing. Only after the advent of electrophysiological recording techniques in the mid-20th century could a first mathematical framework be laid down, resulting in the discoveries of the two Nobel-Prize-winning scientists, Hodgkin and Huxley [19]. Their work proposes a mathematical model that describes the evolution of the membrane potential and the generation of action potentials. It further suggests that action potentials, i.e. spikes, are the key information primitive of the neural alphabet. However, it has only been within the last three decades that new theories have been proposed that shed some light on the mechanism of information encoding into action potentials [20]. The most plausible encoding mechanisms are rate coding, temporal coding, burst coding, and phase coding. A visual representation of each encoding scheme can be seen in Fig. 2.3.

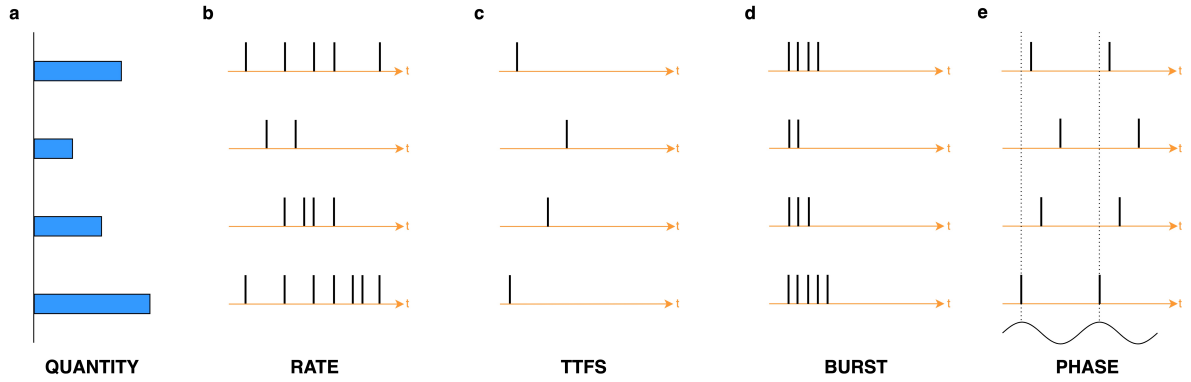


Figure 2.3: Neural coding schemes. (a) represents the real-valued quantity to be encoded in a spike train. This information can be encoded in a spike train using **(b)** rate coding, **(c)** time-to-first-spike coding, **(d)** burst coding, and **(e)** phase coding.

2.2.1. Rate coding

Rate coding is the most widely adopted encoding theory in both neuroscience and spiking neural networks. It encodes information in the spike count, i.e. firing rate, which is proportional to the quantity being encoded. This behavior was observed for the first time in the retina by Hubel and Wisel in their pioneering work [21]. This work demonstrates that if a photoreceptor cell in the retina is stimulated with bright light, it generates a greater number of action potentials transmitted to the visual cortex compared to its response under lower luminosity. This mechanism requires a temporal observation window to count the number of spikes generated by each neuron.

2.2.2. Temporal coding (TTFS)

Temporal coding or time-to-first-spike (TTFS) coding, encodes information in the spike time. For example, as depicted in Fig. 2.3c, a large quantity is encoded in an early spike time, while a small quantity is encoded in a late spike time. This encoding mechanism is particularly suitable for nervous systems that require a very fast response. In fact, temporal coding is the mechanism of communication observed in tactile afferents that requires responses within a few milliseconds [22].

2.2.3. Burst coding

As the name suggests, burst coding encodes information in a spike burst at a specific time. Specifically, this coding scheme has been shown to convey information through the number of spikes or the interval between spikes [23]. Not only the location of the burst can convey information, but also the number of spikes and their relative spacing within a burst [24].

2.2.4. Phase coding

In phase coding, the information is encoded in the phase of the spike with respect to a global oscillation. This distinctive phenomenon has been observed within specific areas of the brain, such as the hippocampus and the olfactory system [25]. The oscillations within the neuron population of these regions can serve as a reference signal for the activity of other neurons. For example, O'Keefe has experimentally demonstrated that, in a rat's brain, the phase of a spike that occurs with an oscillation of the neural population of the hippocampus conveys information about the rat's spatial location [26].

2.2.5. Comparison and discussion

Each coding scheme has different implications for software and hardware implementations. In the context of hardware-software co-design, these implications hold significant potential to facilitate the development of hardware architectures with reduced power consumption and fast inference time. In 2021, Guo et al. [11] propose a systematic review of each coding scheme using metrics such as accuracy, latency, number of synaptic operations (# SOP), area, power, and resilience to quantization. The authors utilize Fashion-MNIST as an evaluation dataset for each scheme. The qualitative results of each coding scheme for each metric obtained through this study are summarized in the heat map of Fig. 2.4.

The accuracy, latency, # SOP, and resilience to quantization metric were obtained from a software simulation of each scheme. Conversely, the area requirements were obtained by determining the equivalent number of NAND gates for each scheme. Finally, for power estimations, each coding scheme was implemented on a Xilinx VC709 FPGA board running with a clock frequency of 100 MHz. The numerical results obtained from these measurements were first normalized using a min-max normalization method and then rated to generate a table similar to the proposed heatmap. It is evident that TTFS encoding provides the most favorable trade-offs among the different schemes.

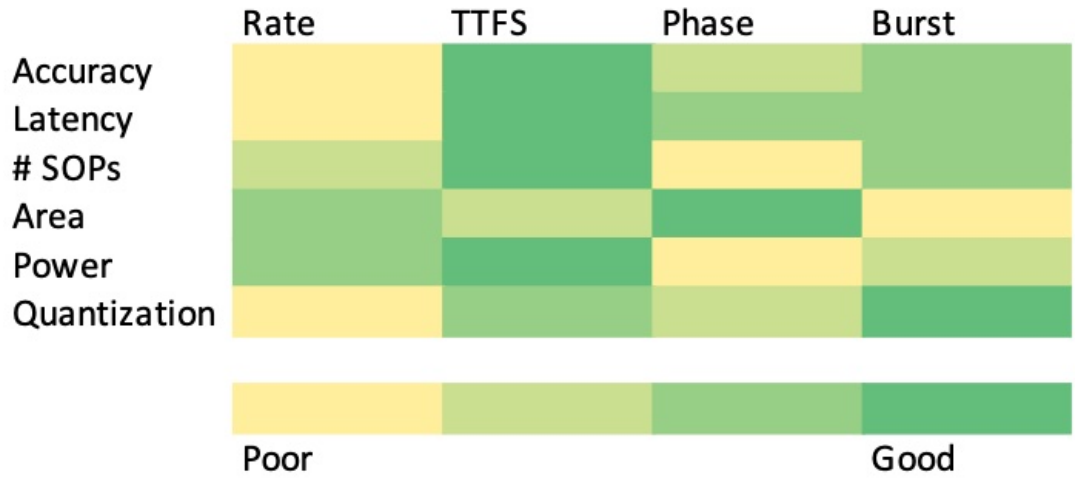


Figure 2.4: Neural coding comparison. Qualitative heatmap of the different encoding schemes evaluated for accuracy, latency, number of synaptic operations (# SOPs), area, power and resilience to quantization. The heatmap is generated form Table 9 of [11].

To gain insights into the main advantages related to latency and power efficiency, which are key metrics for our research question, it is valuable to examine the standard machine learning problem of image classification. Generally, as will be discussed in Section 2.4, networks of spiking neurons are organized in a layered structure where the output layer contains as many neurons as classes of images. The objective of the network is to increase the activity of the output neuron assigned to a specific class when the network is presented with an image of that class.

In the context of rate coding, each neuron in the output layer needs to be observed for the same duration before concluding which one is the most active, and hence determining the classification result. On the contrary, temporal coding eliminates the need to wait for spikes from all output neurons. By encoding the information in the spike time, the correct class can be precisely determined in the instance where the first neuron emits a spike. For this reason, temporal encoding provides a shorter inference time compared to rate coding. Furthermore, in this coding methodology, each neuron is allowed to spike at most once. Therefore, if properly designed, physical neurons in a custom hardware implementation could be gated from processing new incoming data after the occurrence of a spike, thus reducing power consumption. As a consequence, temporal coding has significant potential for designing custom accelerators with fast inference and low power.

2.3. Spiking neuron models

From a biophysical perspective, action potentials are the result of the flow of ions (calcium, potassium, and chlorine) through the membrane. In computational neuroscience, the generation of action potentials can be described by conductance-based or phenomenological models [25]. The former aims at capturing the biophysical mechanism of ions flowing across the neuron's membrane. The latter attempts to capture the essential behavior of the neuron without accounting for the underlying ion channel dynamics. Phenomenological models trade off the simplicity of the mathemati-

cal model for computational efficiency, sacrificing the biophysical details captured by conductance-based models. As a result, to allow for efficient large-scale simulations and hardware implementations, spiking neurons are usually implemented using phenomenological models. The most commonly employed model is the ubiquitous leaky integrate-and-fire (LIF) neuron model. However, the LIF model is a specific case of the more general spike response model (SRM) proposed by Gerstner in [27]. In the incoming sections, the reader will be introduced to both models.

2.3.1. Leaky-integrate-and-fire model

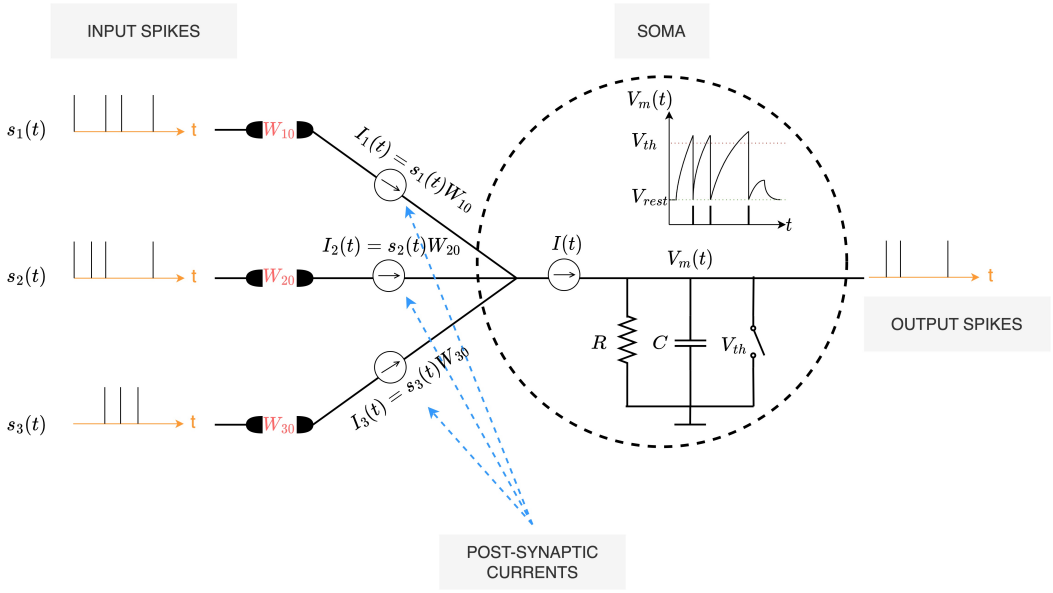


Figure 2.5: Leaky-integrate-and-fire (LIF) model. The figure represents one post-synaptic neuron connected to three pre-synaptic neurons. Each pre-synaptic neuron sends a spike train, e.g. $s_1(t)$, $s_2(t)$ and $s_3(t)$, through the synapses with weights W_{10}, W_{20} and W_{30}. These spike trains are converted into currents by the synapses and are summed up into the current $I(t)$. This current goes through an RC circuit where part of it is integrated in the capacitor C and another part is leaked through R. The switch represents the resetting mechanism of the neuron. If the membrane potential $V_m(t)$ crosses the *threshold voltage* V_{th}, the switch is closed and the capacitor is discharged, thereby resetting the membrane potential to its resting potential.

The LIF model describes the behavior of the membrane potential $V_m(t)$ and the generation of spikes by incorporating the idea of leaky integration and threshold crossing. A graphical representation of the model is provided in Fig. 2.5. The model consists of an RC circuit driven by the current $I(t)$, which results from the sum of all post-synaptic currents obtained by multiplying the input spike trains $S_i(t)$ with the associated synaptic strength W_{ij}, where i and j represent the pre-synaptic and post-synaptic indices, respectively. It is worth noting that the former is a simplified description of the dynamics[2]. The differential equation describing the behavior of a LIF model is

$$\tau \frac{dV_m(t)}{dt} = -V_m(t) + I(t)R, \tag{2.1}$$

[2]After the generation of a spike, the neuron goes into an hyperpolarization phase, also called refractory period, where it is insensitive to incoming signals, which is not modelled here.

where $\tau = RC$ is the time constant of the model. The solution of Eq. (2.1) for constant input current is

$$V_m(t) = IR + [V_{rest} - IR]e^{-\frac{t}{\tau}}. \tag{2.2}$$

From this solution, it becomes evident that, at $t = 0$, the membrane voltage is at the resting voltage V_{rest} and exponentially increases to a steady-state voltage IR with a time constant of τ. By approximating Eq. (2.2) with the forward Euler method and incorporating the resetting mechanism, the evolution of the membrane potential of the LIF model can expressed in a time-stepped equation as

$$V_m[t] = \beta V_m[t-1] + W_{ij}S[t] - S_{out}[t-1]V_{th}, \tag{2.3}$$

where β is a parameter that represents the exponential decay $e^{-\frac{t}{\tau}}$ of the membrane potential, $W_{ij}S[t]$ is the post-synaptic current, i.e. $I(t)$, and $S_{out}[t-1]V_{th}$ the reset mechanism. In any generic spiking model, the membrane potential is constantly compared to the *threshold voltage* Vth as formulated in Equation 2.4. Here, $S_{out}[t]$ is the spike event expressed as

$$S_{out}[t] = \begin{cases} 1, & if \;\; V_m[t] > V_{th} \\ 0, & otherwise. \end{cases} \tag{2.4}$$

It is worth highlighting that different reset mechanisms are also possible. Here, we opt for a *soft reset*, where V_{th} is subtracted from the membrane potential when the neuron spikes.

2.3.2. Spike response model (SRM)

The spike response model (SRM) is a more generic model that attempts to capture the temporal dynamics of a neuron in response to an incoming spike. While the mathematical formulation of a LIF model is typically described by differential equations, the SRM describes the time evolution of the membrane potential as an integration over previous time intervals [27].

For example, consider a post-synaptic neuron which receives a spike from one of the I pre-synaptic neurons at time t_i and generates a spike at time t_j. The SRM describes the evolution of the membrane potential as a response to the incoming spike in accordance with Eq. (2.5), where the function ϵ is a kernel that describes the post-synaptic potential (PSP) after the occurrence of a spike from a pre-synaptic neuron, and where the function η, known as the refractory kernel, describes the neuron's resetting mechanism.

$$V_m[t] = \sum_{i}^{I} W_{ij}\epsilon[t - t_i] - \eta[t - t_j] \tag{2.5}$$

It is worth highlighting that the SRM makes the dynamics that governs the timing of the spike explicit. For example, Eq. (2.5) can be written in terms of the post-synaptic spike time, i.e. t_j, by setting $V_m(t) = V_{th}$.

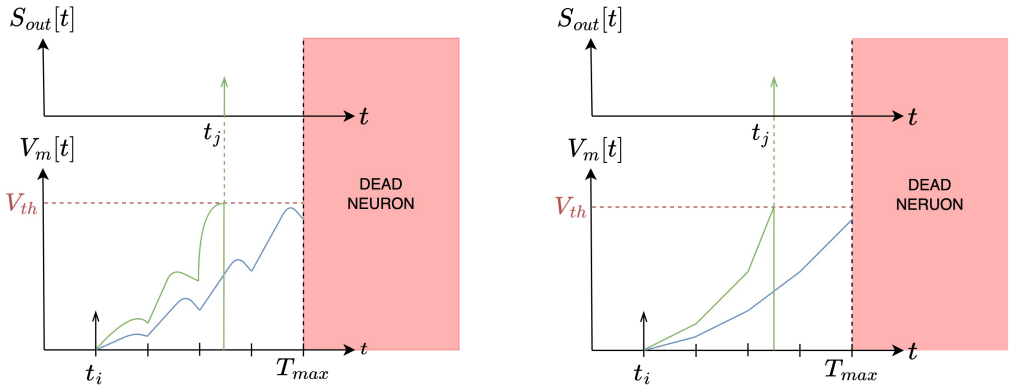
SRM with Alpha-PSP Kernel

Different PSP kernels are viable, among which the alpha function, defined as

$$\epsilon[t] = \frac{t}{\tau} e^{[1-\frac{t}{\tau}]}, \tag{2.6}$$

is the one that most closely resembles the shape of post-synaptic potentials observed in biological neurons [28, 29]. The alpha function describes the evolution of the membrane potential as a rapid initial increase followed by an exponential decay with a time constant of τ. An example of the evolution of the membrane potential following a pre-synaptic spike with an alpha kernel is provided on the left side of Fig. 2.6.

It is important to highlight that the utilization of an exponential function in the definition of the kernel presents considerable challenges when implemented in hardware [30]. For instance, exponential functions cannot be directly computed in hardware; instead, various algorithms can be employed, differing in convergence speed, complexity, and resource utilization, to approximate the exponential behaviour. Hence, the hardware implementation of an exponential function requires a careful study of the trade-off between the different algorithms based on the specific hardware requirements.



(a) Alpha PSP **(b)** Rel PSP

Figure 2.6: Membrane potential evolution of Alpha PSP and Rel PSP. (a) Alpha post-synaptic potential (PSP) and **(b)** Rel PSP. Green lines represent the evolution of the membrane potential for large weights and blue lines for small weights. If the weights are too small, a post-synaptic neuron might spike after the simulation time window, i.e. T_{max}, thereby generating a *dead neuron*.

SRM with Rel-PSP Kernel

The Rel-PSP kernel, proposed by Zhang et al. in their work [31], is defined as

$$\epsilon[t] = \begin{cases} t - t_i, & if\ t > t_i \\ 0, & otherwise. \end{cases} \tag{2.7}$$

The evolution of the membrane potential for a Rel-PSP kernel in response to a pre-synaptic spike at t_i is depicted in Fig. 2.6b. As one can see, the Rel-PSP kernel has a linear response to a pre-synaptic spike similar to a Relu function. It works by linearly integrating the corresponding synaptic strength W_{ij} after the occurrence of a

pre-synaptic spike, i.e. t_i. If the membrane potential crosses the threshold voltage V_{th}, an output spike is emitted. The spike time can be calculated by setting $V_m[t] = V_{th}$ in Eq. (2.5) and Eq (2.7) resulting in the post-synaptic spike time

$$t_j = \frac{V_{th} + \sum_i^I W_{ij} t_i}{\sum_i^I W_{ij}} \quad if\ t_j > t_i. \tag{2.8}$$

It is important to highlight that the kernel function defined in Eq. (2.7) does not contain any exponential function, thereby relaxing the hardware resource requirements compared to an Alpha-PSP kernel.

2.4. Spiking neural networks (SNNs)

In the previous section, we provided an overview of two major computational models of a spiking neuron. However, it is fundamental to understand that individual neurons possess limited computational capabilities. Therefore, in the following sections, we will introduce the reader to the conventional organizational structure of a neural network and the training methodology of this structure. For the latter, we will provide a general description of the back-propagation algorithm. Furthermore, we will discuss how this algorithm can be employed in a TTFS-encoded network.

2.4.1. General network architecture

Inspired by the layered architecture of biological neurons, networks of spiking neurons are conventionally organized in a layered structure as represented in Fig. 2.7.

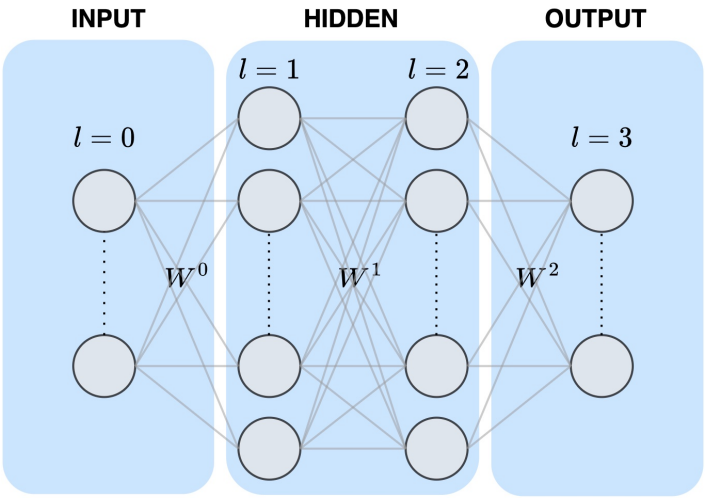


Figure 2.7: Layered architecture. Example of a multilayered spiking neural network with two hidden layers, i.e. $l = 1$ and $l = 2$, one input layer, i.e. $l = 0$, and one output layer, i.e. $l = 3$, where l is the layer index. The number of neurons in each layer and the number of hidden layers is adjustable. For the purpose of simplicity, in this figure, we only show two hidden layers. This organization allows for hierarchical processing of information: the input layer represents the signal from the outside world, e.g. an image of a digit, the hidden layers represent the computational elements, and the output layer represents the final response of the network, e.g. the classified digit. The interconnection between neurons of subsequent layers, represented by the grey lines, are the synaptic strengths, conventionally represented by the symbol W^{l-1}.

It is worth highlighting that the input layer does not perform any computation and it only serves as a gate to the external world. For example, if the input to the network is the image of a digit, each neuron of the input layer is the value of a pixel of the image.

In a network of spiking neurons, the output of a given layer, i.e. $y^l[t]$, is a function of the outputs of the previous layer, i.e. $y^{l-1}[t]$, and the synaptic strength W^{l-1}. This relationship can be expressed as

$$y^l[t] = g[y^{l-1}[t], W^{l-1}], \tag{2.9}$$

where $y^l \in \mathbb{R}^J$, $y^{l-1} \in \mathbb{R}^I$ and $W^{l-1} \in \mathbb{R}^{I \times J}$, with J being the total number of post-synaptic neurons in layer l, and I being the total number of pre-synaptic neurons in layer $l-1$. In Eq. (2.9) the function $g^l[\cdot]$ models the dynamics of the neuron. For example, $g^l[\cdot]$ can express the neural dynamics defined by a LIF or SRM model.

2.4.2. General training

Networks of neurons are conventionally trained by coupling stochastic gradient descent (SGD) [8] with back-propagation (BP) [7]. The former iteratively adjusts the strength of the synaptic connections, with the aim of reducing the error between the network output and the target output, by minimizing a loss function L. For instance, according to SGD, the new value of a synaptic connection that minimizes the loss function, i.e. $\tilde{W}_{ij}$, can be calculated as

$$\tilde{W}_{ij} = W_{ij} - \eta \frac{\partial L}{\partial W_{ij}}, \tag{2.10}$$

where η is the learning rate, and $\partial L / \partial W_{ij}$ is the gradient of the loss function L, with respect to the synaptic strength parameter W_{ij}, found with the back-propagation algorithm, as illustrated in Fig. 2.8.

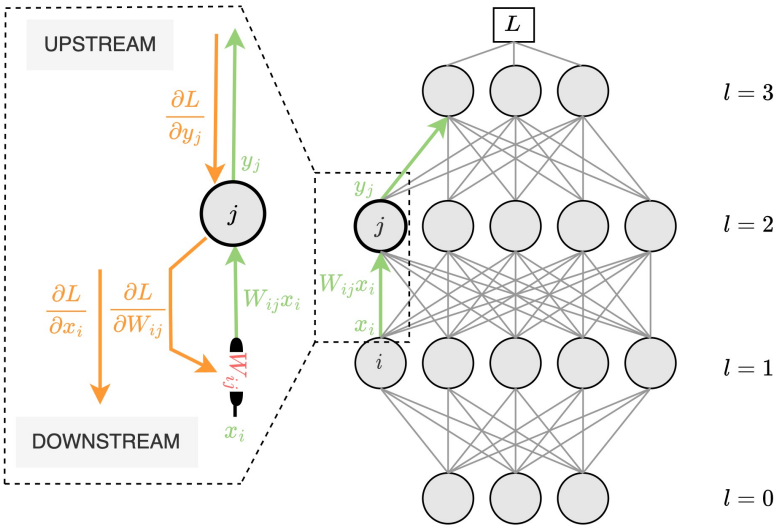


Figure 2.8: Back-propagation. Computational graph for the back-propagation algorithm. Green lines represent the forward propagation of inputs and the orange lines the back-propagation of gradients.

This algorithm generally comprises two phases: the *forward propagation* and the *backward propagation* [7]. The former propagates the input signals through the network and evaluates the error of the network's output with respect to the target output. The latter involves the calculation of gradients of the loss function with respect to the weights of the network, i.e. $\partial L/\partial W_{ij}$. These gradients indicate the direction of steepest descent that minimizes the loss function. In other words, they indicate how the parameter W should change to decrease the loss. Figure 2.8 illustrates the computational graph of the back-propagation algorithm required to update the parameter W_{ij} that connects the post-synaptic neuron j with the pre-synaptic neuron i.

In the forward propagation, the input signal x_i is multiplied by the synaptic strength parameter W_{ij} and processed by the neuron to produce the output signal y_j. In the backward propagation, the neuron receives the upstream gradient, i.d. $\partial L/\partial y_j$, from the layer above, which indicates how the output of the given neuron influences the loss function L. By applying the chain rule, the algorithm finds the direction of steepest descent that minimizes the loss as

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial y_j}{\partial W_{ij}} \frac{\partial L}{\partial y_j}. \tag{2.11}$$

In addition, the upstream gradient, i.e. $\partial L/\partial y_j$, is used together with $\partial y_j/\partial x_i$ to find the downstream gradient $\partial L/\partial x_i$ as

$$\frac{\partial L}{\partial x_i} = \frac{\partial y_j}{\partial x_i} \frac{\partial L}{\partial y_j}. \tag{2.12}$$

This gradient becomes the upstream gradient for the neuron in the previous layer, i.e. neuron i, thereby enabling subsequent application of the chain rule. In other words, to compute the gradient of the loss function for the synaptic strength parameter W_{ij} of each layer, the chain rule is iteratively applied from the output to the input layer.

It is of fundamental importance to understand that, up to this point, we have considered the signal propagating through the network as a real-valued number. However, spiking neural networks (SNNs) process spike trains, denoted as $s(t)$. To account for the effect of time, the computational graph is unrolled through time, as illustrated in Fig. 2.9. The effect of each synaptic strength parameter on the loss function at each time step is summed up together as per Eq. (2.13), where T_{max} is the duration of the spike train. Upon completing the sequence processing, i.e. $t = T_{max}$, the gradient of Eq. (2.13) is propagated backwards across different time steps and layers to update the parameter W_{ij}. This process is illustrated in Fig. 2.9, and takes the name of error *back-propagation through time (BPTT)*.

$$\frac{\partial L}{\partial W_{ij}} = \sum_{t}^{T_{max}} \frac{\partial L[t]}{\partial W_{ij}} \tag{2.13}$$

Additionally, spiking neurons require a non-differentiable threshold function to generate output spikes as per Eq (2.4). Since the chain rule requires each pathway to be differentiable, the non-differentiability of the spiking function hinders the applicability of the back-propagation algorithm. Historically, various techniques have been adopted to address the non-differentiable nature of spiking neurons [10]. The most robust and popular solution is to use a *surrogate gradient* of the threshold function

during backward propagation [32]. This approach has proven to be highly effective and has enabled spiking neural networks to achieve state-of-the-art results in various tasks.

Another solution consists in applying the back-propagation algorithm to the spike time [33]. In other words, instead of calculating the gradients through the discontinuous spike generation mechanism, gradients are calculated with respect to the spike time, which is a continuous function as long as a spike occurs [33]. In TTFS-encoded SNNs, the information is encoded in the spike time, which allow sidestepping the problem of non-differentiable threshold functions.

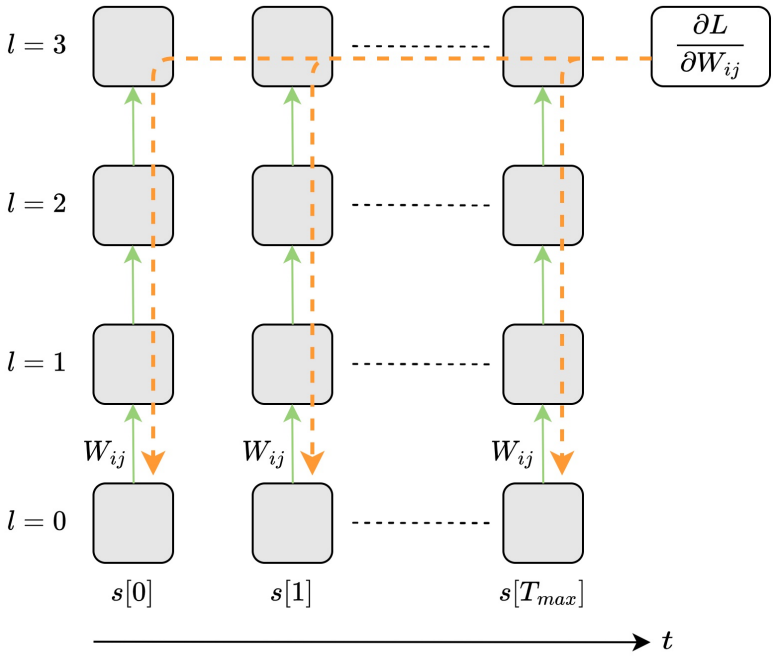


Figure 2.9: Back-propagation through time (BPTT). Computational graph of the BPTT algorithm. Green lines represent the forward propagation of inputs while the orange lines the back-propagation of error gradients across layers and time. Differently from the normal BP algorithm, to process a spike train, BPTT requires making a copy of the network for each time step.

Least but not last, to evaluate the performance of a neural network, the *dataset*, which contains multiple labeled examples of a given problem, is typically divided into two distinct subsets: the *training set* and the *test set*. The first, is employed to train the network, e.g. by applying SGD and BP, whereas the second is used to verify that the network can *generalize* to unseen data. In the context of a classification problem, the performance of a model is evaluated using the accuracy metric defined as

$$Accuracy = \frac{N_P}{N_T}, \tag{2.14}$$

where N_P is the total number of correct predictions of the network, and N_T is the total number of examples in the test set. A high accuracy metric on the test set reflects the network's ability to properly generalize to unseen data.

2.4.3. Training the SRM with Rel-PSP Kernel

Similarly to Fig. 2.8, the computational graph for the back-propagation algorithm applied to a SRM with a Rel-PSP kernel using TTFS coding is shown in Fig. 2.10.

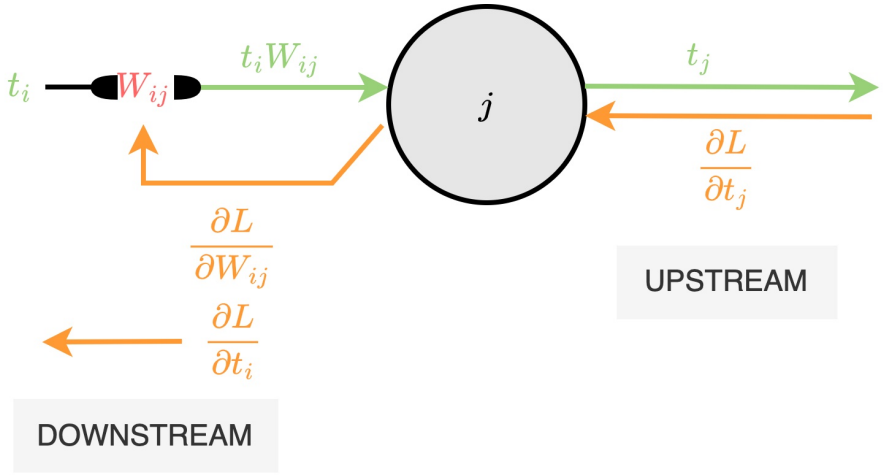


Figure 2.10: Rel-PSP computational graph. The green lines represent the forward propagation path, and the orange ones the backward propagation path. The trainable parameters are colored in red, e.g. W_{ij}. The spike times of neurons i and j of Fig. 2.8 are t_i and t_j, respectively. The synaptic strength parameter, e.g. W_{ij}, is updated with the gradient $\partial L/\partial W_{ij}$. The gradient $\partial L/\partial t_j$ is the upstream gradient from the next layer while $\partial L/\partial t_j$ is the upstream gradient to the previous layer.

Considering Eq. (2.5), together with Eq. (2.7) and Eq. (2.4), it is possible to find the back-propagation gradient required to update the synaptic strength parameter W as

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial t_j}{\partial V[t_j]} \frac{\partial V[t_j]}{\partial W_{ij}} \frac{\partial L}{\partial t_j} = \left(\frac{-1}{\sum_i^I W_{ij}} \right) (t_j - t_i) \frac{\partial L}{\partial t_j}, \tag{2.15}$$

and the downstream gradient as

$$\frac{\partial L}{\partial t_i} = \frac{\partial t_j}{\partial V[t_j]} \frac{\partial V[t_j]}{\partial t_i} \frac{\partial L}{\partial t_j} = \left(\frac{-1}{\sum_i^I W_{ij}})(-W_{ij} \right) \frac{\partial L}{\partial t_j} \tag{2.16}$$

The loss function L employed by Zhang in [31], is a regular cross-entropy function which attempts to reduce the time of the spike of the target neuron and, concurrently, extend the time of the spike of non-target neurons. In the context of a classification task, the output layer of a spiking network, e.g. $l = 3$ in Fig. 2.8, consists of multiple neurons, each of which spikes at a different time. The output neuron that indicates the correct class must have the shortest spike time. To achieve this goal, the loss function is defined as

$$L = -ln \frac{exp(-t_{\hat{j}})}{\sum_j^O exp(-t_j)}, \tag{2.17}$$

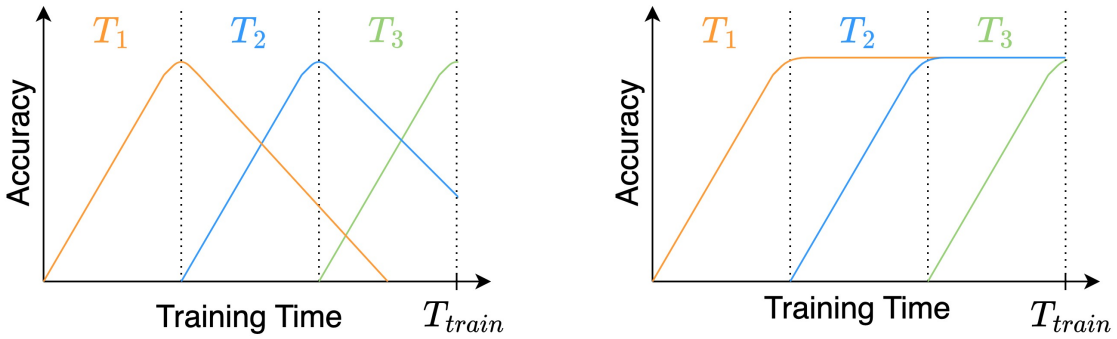
where O is the number of neurons in the output layer, j their index, and $\hat{j}$ the index of the neuron that represents the correct class.

2.5. Continual learning (CL)

Humans have the innate ability to continuously learn new tasks without forgetting old ones. This ability is conventionally called *continual learning* in the deep learning community. On the contrary, spiking and artificial neural networks cannot learn new tasks sequentially without forgetting previous ones. In other words, both SNNs and ANNs suffer from the problem of *catastrophic forgetting*. This section provides the reader with a formal definition of the problem of catastrophic forgetting and presents various solutions proposed in the existing literature.

2.5.1. Problem statement

In a typical continual learning setup, a neural network must sequentially learn the tasks $T = (T_1, T_2, .., T_n)$ without catastrophically forgetting any of the previously learned tasks. This condition drastically differs from the typical neural network training procedure, where examples of different tasks are independently sampled from the training set and presented to the network in an interleaved fashion and all trained at the same time. Indeed when tasks are trained sequentially, as illustrated in Fig. 2.11a, the accuracy of the network for each task on the test set tends to decrease when new tasks are introduced. In contrast, the main goal of continual learning is to minimize the amount of accuracy drop across multiple tasks when trained sequentially as illustrated in Fig. 2.11b.



(a) Catastrophic Forgetting

(b) Ideal Continual Learning

Figure 2.11: Continual learning accuracy. In both figures, three tasks are presented sequentially, where the start of training for each new task is represented by the dashed lines. In **(a)**, as a new task is presented to the network, the accuracy of previously learned task drops, demonstrating catastrophic forgetting. In **(b)**, which is the ideal scenario, the accuracy of previously learned tasks does not drop as new tasks are learned. In both figures T_{train} defines the last training step of the last task.

In addition, as Hsu et al. suggested in [34], the difficulty of addressing the problem of catastrophic forgetting varies depending on experimental protocols. For example, the author identified three different scenarios of increasing difficulty based on the presence of a task ID to identify the change in task during training. These scenarios are:

- **Incremental Task Learning:** the task ID is provided,
- **Incremental Domain Learning:** the task ID is inferred,

- **Incremental Class Learning:** the task ID is neither provided nor inferred.

Furthermore, in a typical continual learning environment, the network architecture can be *single-headed* or *multi-headed* [34]. In a single-headed architecture, all parameters, i.e. Ws, are shared between different tasks, while in a multi-headed architecture, the output layer has a separate set of parameters for each task. In this thesis, we will focus on the simplest case of incremental task learning with a multi-headed architecture. To evaluate the performance of the proposed solution, the average accuracy metric, as per Eq. (2.14), is calculated across different tasks at the end of the sequential training, i.e. T_{train}.

2.5.2. Evaluation datasets

In the domain of ANNs, the effectiveness of a proposed solution in mitigating the problem of catastrophic forgetting is conventionally assessed with the Split-MNIST [35] and the Permuted-MNIST [36] datasets, both derived from the original MNIST dataset of handwritten digits [37]. In the case of Split-MNIST, the dataset is divided into distinct subgroups, resulting in binary classification tasks, as illustrated in Fig. 2.12a. For instance, Task 1 involves distinguishing between digits 0 and 1. On the other hand, Permuted-MNIST generates tasks by permuting pixels within MNIST images, as shown in Fig. 2.12b. Essentially, each task corresponds to a specific permutation applied to every image in the dataset.

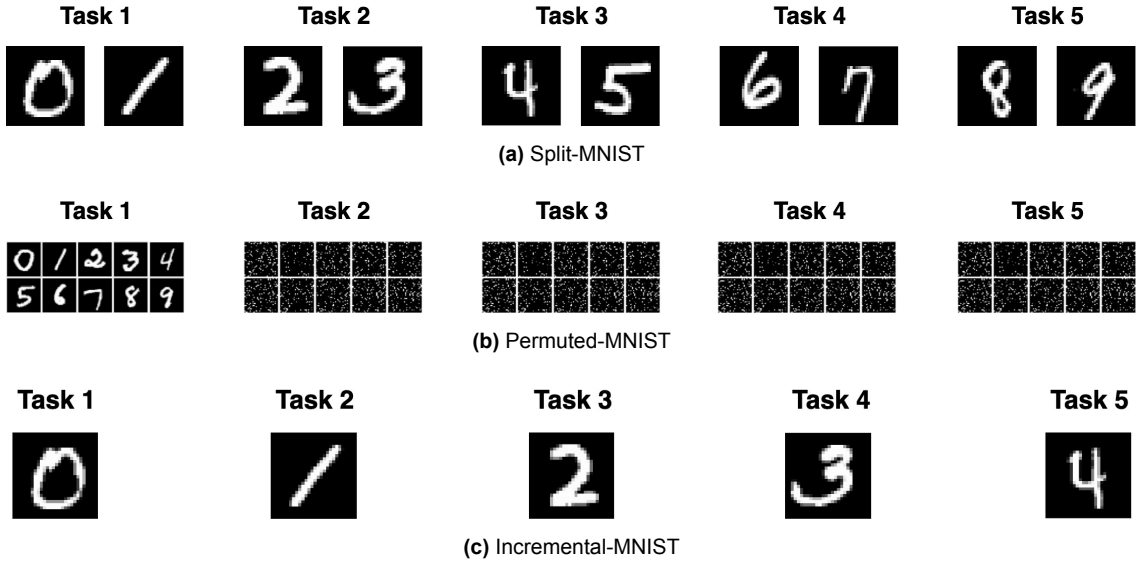


Figure 2.12: Datasets for continual learning. In **(a)**, the MNIST dataset is partitioned into distinct binary classification subgroups. In Task 1, the network is exposed to multiple images of digits 0 and 1 during training, with the goal of classifying whether an image belongs to one class or the other. In **(b)**, each digit in the MNIST dataset is permuted to generate a new task. The network is then presented with multiple examples of permuted digits, spanning from 0 to 9, and its objective is to correctly classify which digit the image represents. In **(c)**, each digit of the MNIST dataset is fed to the network incrementally. For example, in Task 1, only the images of digit 0 are presented to the network. The objective of the network is to correctly classify which digit the image represents.

Furthermore, in the neuromorphic community, a commonly employed approach is to incrementally classify the digits of the MNIST dataset [38, 39, 40]. This idea

is depicted in Fig. 2.12c, where, during a specific task, the network is exposed to multiple images of a single digit with the objective of classifying the digit against the others. Indeed, it is crucial to highlight that, as a new digit (or task) is presented to the network, images of previous digits are no longer supplied, thus establishing a typical scenario for continual learning.

2.5.3. Review of solutions

This section provides a review of the literature, examining various approaches and strategies to mitigate catastrophic forgetting in both the ANNs and SNNs worlds. This review seeks to uncover a biologically inspired strategy that could be implemented in a TTFS encoded neural network trained with back-propagation. Hence, only works that are biologically inspired and employ the back-propagation algorithm are reported.

Strategies used to mitigate the problem of catastrophic forgetting usually fall into three distinct categories: regularization-based, which attempts to prevent changes in parameters that are important for a given task as a new task arrives; architecture-based, which adds a set of parameters for each new task; and memory-based, which consists of feeding a few examples from previous tasks as the new task is being trained. The most relevant works addressing this problem are chronologically listed in Table 2.1.

Author	Year	Strategy	Type
		ANNs	
Kirkpatrick [41]	2017	Elastic Weight Consolidation (EWC)	Regularization
Zenke [35]	2017	Synaptic Intelligence (SI)	Regularization
Masse [42]	2018	Context Gating + SI / EWC	Regularization, Architectural
Iyer [13]	2022	Active Dendrites	Architectural
		SNNs	
Hammouamri [43]	2022	Threshold Modulation	Architectural

Table 2.1: Summary of continual learning strategies using back-propagation in ANNs and SNNs.

Within the domain of ANNs, the two most popular approaches to address the problem of catastrophic forgetting are elastic weight consolidation (EWC) and synaptic intelligence (SI). Both approaches introduce a quadratic regularization term in the loss function with the aim of capturing the relevance of each weight to a given task. The former estimates the significance of individual weights by assessing their impact on the network's output after training on each task. The latter estimates the importance of each weight by analyzing how the gradient of the loss function relates to the updates of these weights while training. Specifically, both strategies aim at penalizing changes in weights that are important for previously learned tasks, preventing the overwriting of previously acquired information, and are thus regularization-based techniques.

Although both approaches demonstrated the ability to retain information from previously learned tasks, the work of Masse [42] shows that context gating, coupled with SI or EWC, results in even better information retention. Specifically, when these strategies are benchmarked in a continual-learning environment comprising the sequential learning of 100 tasks from the Permuted-MNIST dataset, EWC achieves a mean test accuracy of $\approx$ 70%, SI of $\approx$ 80% and Context Gating + SI / EWC of $\approx$ 95%. The

solution proposed by Masse consists of projecting a context vector onto the neurons of a hidden layer to gate a set of non-overlapping neurons for each task, as depicted in Fig. 2.13 for tasks 3 and 5, with a one-hidden-layer network. The hidden layer receives a context vector that gates different neurons for different tasks. In this context, gating means that the context vector inhibits the output of each selected neuron. Furthermore, synaptic connections are updated only for neurons that are not gated, thus suppressing the interference of the current task on previously learned ones. This approach is inspired by the experimental evidence that areas of the brain, such as the prefrontal cortex, generate gating signals to the cortical area, allowing for task-dependent processing of information [42].

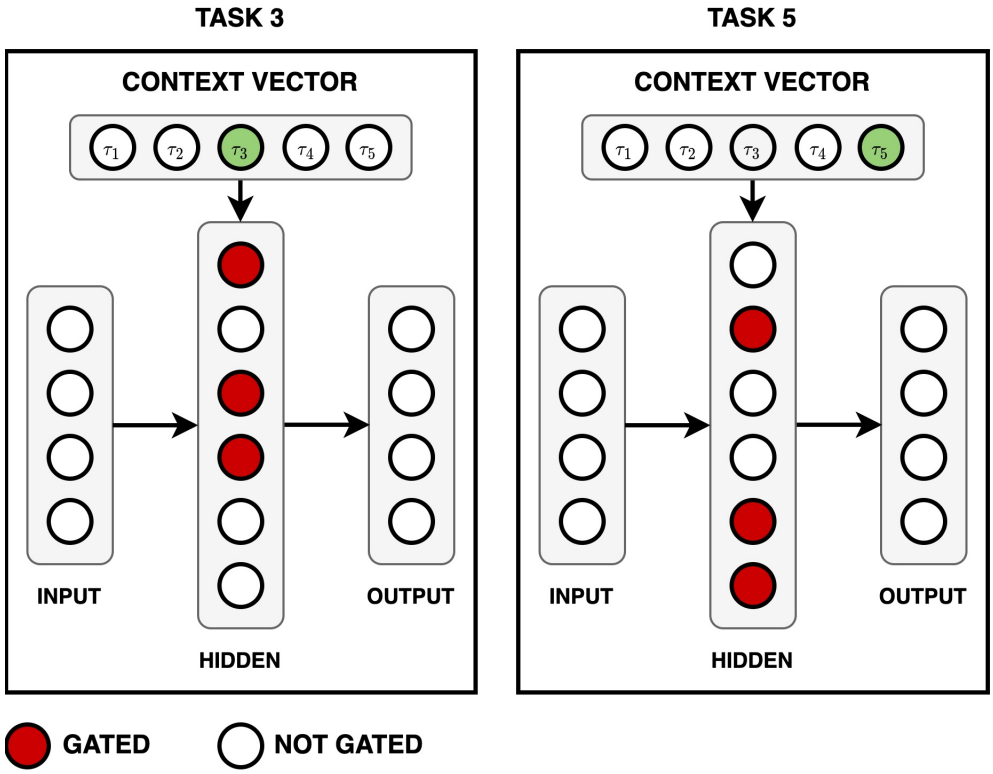


Figure 2.13: Context gating. The neurons in the hidden layer receive a different context vector for each task. This context vector gates a different non-overlapping subset of neurons for each task.

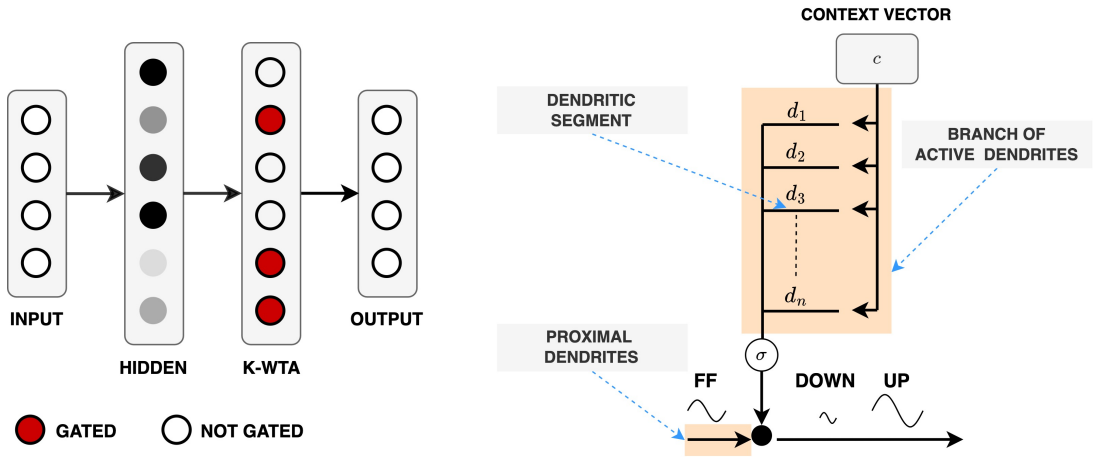
The concept proposed by Masse is extended even further by Iyer et al. in their work [13]. Specifically, the authors hypothesize that the gating mechanism is controlled by the active dendrites of pyramidal neurons. This hypothesis takes inspiration from the concept that a dendritic spike initiated by active dendrites can depolarize the neuron for a long period of time, typically on the order of seconds [44]. This depolarization makes the neuron more likely to spike as new inputs arrive at the proximal dendrites, thus creating a modulation effect on the cell response.

It should be emphasized that conventional artificial or spiking neuron models are anachronistically rooted in the perceptron model [45]. This model assumes a linear effect of all synapses on the cell, as postulated in the *point neuron* model proposed by Lapicque [46]. This behavior is particularly prominent in proximal dendrites compared to their distal counterparts, i.e. active dendrites, which, in turn, have the largest

number of synaptic connections. Indeed, current neuron models, whether artificial of spiking, typically do not model the effect of active dendrites. As a consequence, Iyer et al. attempt to enhance the regular point neuron by implementing the following key aspects:

- each neuron contains multiple isolated active dendritic segments that handle contextual inputs using distinct sets of weights,
- contextual inputs can either up-modulate or down-modulate the feedforward activation of active dendrites,
- a *k*-winner-take-all (*k*-WTA) layer is used to select the *k* neurons with the highest activation to ensure sparsity.

The neural architecture proposed by Iyer et al. consists of a hidden layer followed by a *k*-WTA layer (Fig. 2.14a), and each neuron in the hidden layer is enhanced with branches of active dendrites (Fig. 2.14b). The effect of feedforward (FF) inputs, represented by the proximal dendrites, is modulated by the active dendrites depending on the response to the context vector.



(a) Network Architecture **(b)** Neuron Model

Figure 2.14: Active dendrites. (a) represents the network architecture comprised of input, hidden, *k*-WTA, and output layers. **(b)** represents the enhanced neuron model used in each neuron of the hidden layer.

Consider, for example, the input vector x to the hidden layer, the matrix W representing the synaptic strengths between the input layer and the hidden layer, and the bias vector b. The conventional linear feedforward integration of the input signal performed by proximal dendrites can be expressed as

$$y = Wx + b. \tag{2.18}$$

The branch of active dendrites contains n dendritic segments, where n is the total number of tasks. Each segments is a vector $d_j \in \mathbb{R}^n$. For each dendritic segment, the dot product between the segment vector parameter d_j and the context vector $\vec{c} \in \mathbb{R}^n$ is computed. Subsequently, the segment with the highest activation is selected to generate the dendritic activation d, as per Eq. (2.19).

$$d = \max_j(d_j \cdot c) \tag{2.19}$$

To modulate the neural feedforward activation y, the dendritic activation d is passed through a sigmoid function $\sigma(\cdot)$ that modulates y, resulting in

$$\tilde{y} = \vec{y} \times \sigma(d) = (Wx + b) \times \sigma(\max_j(d_j \cdot \vec{c})). \tag{2.20}$$

As per Eq. (2.20), a positive response to a specific context vector preserves the standard feedforward linear activation, whereas a negative response down-modulates the activation. As a result, neurons relevant for a given task, preserve their activation, whereas neurons that are irrelevant, are gated. Subsequently, the *k*-WTA selects the neurons with the *k* highest activations and gates the rest. In this manner, only a sparse subgroup of the neurons in the hidden layer are activated for a given task.

Furthermore, during the backward pass, only synaptic strengths of winning neurons are updated. Of these winning neurons, only the dendritic branch parameter d_j with the highest response is updated. Iyer et al. hypothesize that following this approach, each dendritic branch learns to detect specific context vectors. Also, since dendritic branches irrelevant to the current task are not updated, modulation of each neuron becomes context-dependent. This, in turn, invokes different subnetworks for each task, thereby reducing interference between tasks and mitigating catastrophic forgetting.

Compared to Masse's solution, the approach proposed by Iyer at al., achieves a 3% lower mean accuracy on the Permuted-MNIST dataset with 100 tasks. However, their solution offers one key advantages: distinct subnetworks automatically emerge from the active dendrites, as opposed to being pre-allocated for each task.

As we have seen up to this point, the problem of catastrophic forgetting has been thoroughly addressed in the domain of ANNs trained with back-propagation. On the contrary, insufficient attention has been directed to the same problem in the domain of SNNs. Part of the reason for this disparity stems from the historical difficulty of applying back-propagation to SNNs, which became feasible only recently [32]. As a consequence, a significant portion of research aimed at addressing the issue of catastrophic forgetting in SNNs [47, 38, 39, 40], employ STDP [48] as the primary learning algorithm, which is known to scale poorly [49].

To the best of our knowledge, the only work that uses a back-propagation-based approach to solve continual learning with SNNs is the work proposed by Hammouamri et al. in [43]. The main idea behind this work is to modulate the threshold voltage of each neuron in a task-dependent manner. To achieve this goal, the authors implement two different networks. One, which is responsible for the conventional classification, and another, which is responsible for modulating the threshold voltage of the first network's output layer, in a task-depend manner. By following this approach, the activity, e.g. number of spikes, is modulated depending on the current task being presented. Indeed, this approach is analogous to the methods proposed in [13] and [42] to mitigate the problem of catastrophic forgetting in ANNs. It is important to note that the network responsible for modulating the threshold values is trained using an evolutionary optimization algorithm. Additionally, the algorithm requires two training steps for each network and has slow convergence rates.

2.5.4. Discussion

This review highlighted the importance of gating neurons in a context-dependent fashion as a fundamental mechanism to alleviate the problem of catastrophic forgetting in both artificial and spiking neural networks. In particular, Iyer suggests that extending the conventional point-neuron model with active dendrites leads to a novel mechanism that, when coupled with a *k*-winner-take-all layer, generates different subnetworks for each task. The generation of these subnetworks is crucial as it prevents any interference between tasks during the training process. For this reason, in Chapter 3 we will investigate if this solution is also applicable to a TTFS-encoded SNN.

2.6. Hardware architectures for event based SNNs

This section serves as a review of the various architectural approaches commonly used for the implementation of event-based digital SNNs. Furthermore, we provide a review of the current digital hardware implementation for continual learning and TTFS encoded networks.

2.6.1. General principles

Hardware architectures of SNNs can be classified into two major categories, the first targeting large-scale applications such as brain simulations or large SNNs, the second targeting small-scale SNNs implementations for embedded neuromorphic computing at the edge. The former focuses on optimizing the design for high throughput and scalability, whereas the latter focuses on optimizing area and power efficiency.

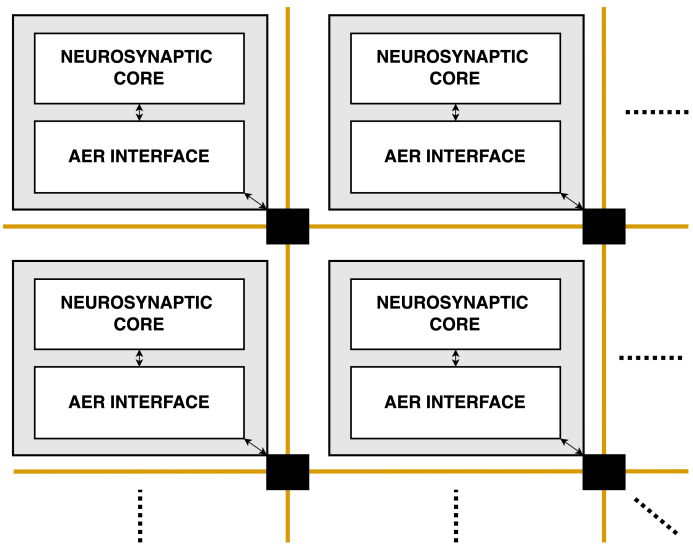
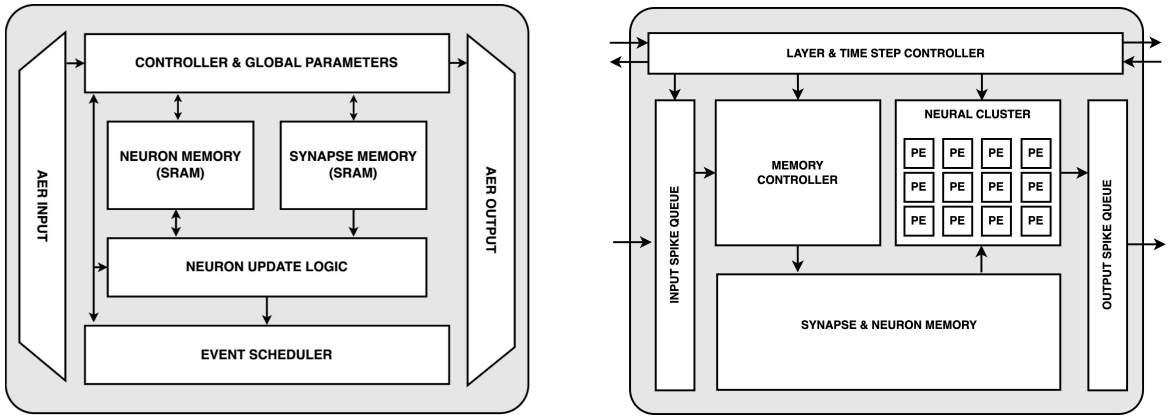


Figure 2.15: Crossbar architecture. This architecture is comprised of a set of neural cores, represented by the grey boxes, interconnected with a network-on-chip, represented by the orange lines. The event-based protocol employed in neuromorphic solutions to exchange spikes between different cores is the address-event-representation (AER) [50]. Each core consists of a *neurosynaptic* core and an AER interface. The first is comprised of local memory and processing elements required to emulate the dynamics of multiple neurons and synapses. The second is comprised of input and output encoder receiving input spikes and dispatching the generated spikes to other neurons through the AER bus, respectively.

Large-scale neuromorphic hardware is conventionally implemented using a crossbar array, as depicted in Fig. 2.15. This architectural design attempts to solve the von Neumann bottleneck by co-locating memory and processing. Examples of systems implementing these architectures include TrueNorth [51] and BrainScaleS [52]. The crossbar architecture is highly scalable as it can be implemented in many core single- and multi-chip configurations.

On the other hand, to optimize for area and power efficiency, small-scale digital implementations can take two approaches: emulating the crossbar architecture by using time-multiplexing [53], as depicted in Fig. 2.16a, or organizing neuron computing elements in a parallel fashion [54, 55, 56, 57], as depicted in Fig. 2.16b. In the first approach, the neurons states in a layer are updated sequentially using shared resources, while in the second approach they are updated in parallel using shared or local resources.



(a) Time-multiplexed architecture

(b) Parallel architecture

Figure 2.16: Small scale architectures. Layer architecture using time-multiplexing and parallel computing. In **(a)** a single computational element updating the state of neurons in a layer using time-multiplexing. The input spikes are sent to the AER input encoder and moved into an event scheduler by the main controller, which then fetches the related synaptic parameters stored in the synapse memory and updates the neuron state. The new state is stored in the neuron memory. Whenever a neuron crosses the threshold, thereby generating a spike, the address of that neuron is sent to the AER output encoder. The interconnection between neurons is specified by the value stored in the synapse memory. In **(b)**, the neural cluster contains multiple instances of a neuron processing element (PE), similar to the neuron update logic of a time-multiplexed architecture. The states and synaptic connections of each physical PE can be stored in a shared memory or in local memory elements.

Consequently, from a layer perspective, time-multiplexed architectures offer advantages in terms of area and power efficiency by sacrificing speed. On the other hand, parallel architectures offer advantages in terms of speed by sacrificing area and power efficiency.

2.6.2. Architectures for TTFS SNNs and continual learning

In this section, we will cover state-of-the-art solutions to two concepts at the core of this thesis: (i) architectures optimized for TTFS coding, and (ii) first proofs-of-concept for continual-learning hardware.

Architectures supporting the TTFS encoding

Frenkel et al. in [58] propose the *spiking online-learning convolutional neuromorphic processor* (SPOON), a digital architecture with online learning, consisting of a convolutional core and a fully-connected core, capable of classifying images from the N-MNIST dataset using a TTFS encoding. Specifically, the fully-connected core contains two layers, one for the hidden neurons and the other for the output neurons, as depicted in Fig. 2.17a. The first utilizes time multiplexing to compute the neural dynamics of 128 neurons, while the second has 10 neurons capable of updating their state in parallel. The proposed architecture can be trained online to achieve 90% accuracy in a single epoch (93% after 100 epochs) and consumes only 655 nJ to classify a sample image (post-layout simulation in a 28nm node). However, SPOON only supports a standard learning setup where all training data is provided in parallel and not sequentially. In other words, this architecture would suffer from catastrophic forgetting in a continual-learning scenario.

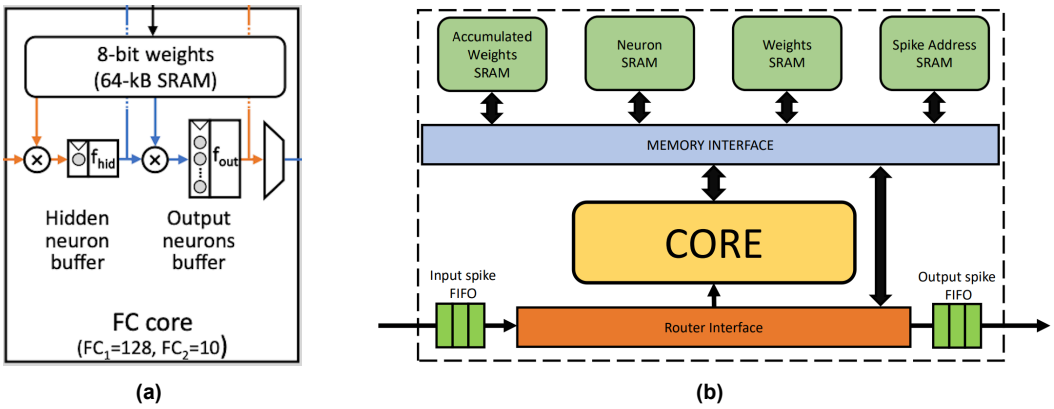


Figure 2.17: TTFS-encoded architectures. In **(a)** the fully connected layer of the SPOON architecture [58], in **(b)** one layer the YOSO architecture [55].

The SRM with Rel-PSP kernel neuron model presented in Section 2.3.2 was successfully implemented in the *you only spike once* (YOSO) digital architecture, as proposed by Srivatsa et al. in [55]. The architecture of a single layer is illustrated in Fig. 2.17b. Each layer contains a single time-multiplexed core that emulates the behavior of 256 neurons. Unlike SPOON, this work does not provide online-learning capabilities. Synthesis results in a 22nm node show that YOSO implementing the Rel-PSP model descibed in [31], is capable of classifying a sample image of the MNIST dataset in 47.71 ms while consuming 0.750 mW, resulting in an energy per inference of 36μJ for a classification accuracy of 98.4%.

In contrast, Widmer et al. implement in [59] the SRM with Rel-PSP kernel neuron model in a novel digital architecture that updates the state of each neuron in parallel. Notably, in this design, the post-synaptic membrane potential is integrated in a local register, thereby reducing the amount of data movement to and from the memory, compared to the previous two designs. This architecture has been tested with the Iris dataset [60], where post-layout simulation results in a 7nm node show that one sample image can be classified in only 21 ns while consuming 2.62 mW, resulting in an energy per inference of 55pJ for a classification accuracy of 96 %, yet restricted to a simple dataset.

In summary, the different existing architectures for TTFS-encoded networks typically use multi-layer architectures. However, while YOSO utilizes a time-multiplexed approach for its computational element, SPOON utilizes a hybrid approach between time-multiplexed and parallel computing of neuron dynamics. Finally, the work of Widmer et al. proposes an architecture where neurons are fully updated in parallel and the membrane potential is stored in a local register.

Architectures supporting continual learning

Regarding hardware architectures that target a continual-learning setup, the available literature is notably limited. The only two works that attempt to address these challenges in a bio-inspired fashion are MetaplasticNET [61] and SCOLAR [62]. Both works provide online continual learning capabilities and implement the neuron's state update using a systolic array of processing elements (PEs). The former uses a 2D systolic array of PEs to update the state of the neuron, while the latter uses a 1D array. The first architecture is designed to implement networks of artificial neurons, whereas the second for implementing networks of spiking neurons. As SNNs are at the core interest of our research question we report only the second architecture as shown in Fig. 2.18. Each processing element sequentially updates the state for $N_{fold} = N_h/N_{pe}$ neurons, where N_h is the number of hidden neurons and N_{pe} is the number of processing elements, providing some level of parallelism.

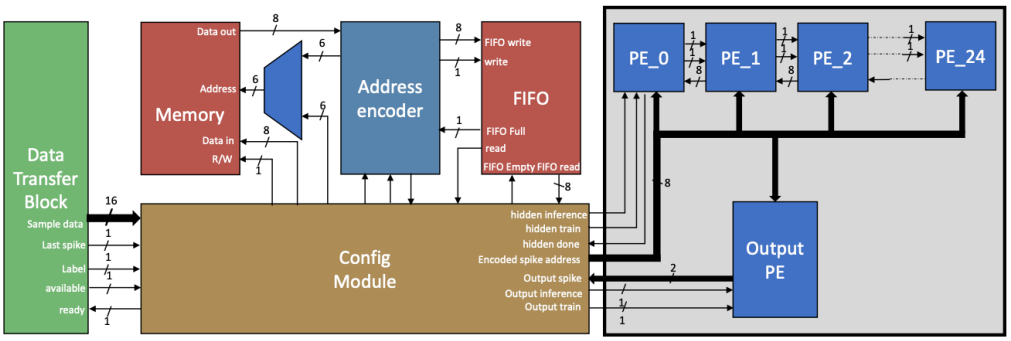


Figure 2.18: SCOLAR architecture. On the right side of the image is the systolic array of processing elements (PE) computing the neuron's updates. Image source from [62].

Discussion

Notably, none of the TTFS architectures support continual learning[3]. Similarly, while different levels of parallelism have been investigated in current continual-learning architectures for efficiency purposes, none of them supports sparse data representation such as the TTFS encoding. This reveals an obvious gap between robust learning and processing efficiency in the current literature, which we will address in Chapter 3.

[3]SPOON is the only architecture supporting learning with a TTFS encoding, but it still suffers from catastrophic forgetting.

3

Design

This chapter, comprising two sections, describes the proposed solution to the main research question. The first section, focusing on software development, explains the proposed neural model enhanced with active dendrites, the network architecture, and the mathematical derivation of the gradients necessary for the back-propagation algorithm. The second section, which focuses on hardware design, explains the general hardware architecture, the neuron model, the neural cluster, and the relative controllers.

3.1. Software design

This section provides a detailed explanation of the mathematical model, the network architecture, and the gradients derivation of the proposed model. The computational model resulting from the proposed solution has been implemented using the conventional *PyTorch* [63] framework with the *Python* programming language. The detailed code implementation can be found here.

3.1.1. Neuron model

To address the TTFS encoding mechanism, the proposed neuron model is based on a SRM with the Rel-PSP kernel introduced by Zhang et al. in [31]. To mitigate the problem of catastrophic forgetting, this kernel is enhanced with an adapted version of active dendrites, as proposed by Iyer et al. in [13]. An illustration of the proposed neuron model is shown in Fig. 3.1a.

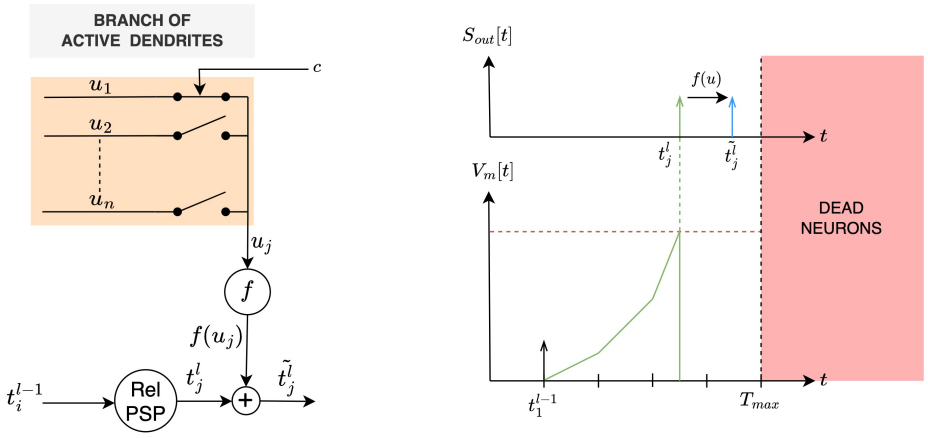
As explained in Section 2.5.3, Iyer et al. introduce a modulation factor of neural activity depending on the response of the active dendritic branch to the context vector c. For of a TTFS-encoded model, the activity of a neuron is represented by the spike time t_j, rather than a real-valued number. Hence, modulating the activity of a TTFS-encoded neuron is equivalent to modulating its spike time.

To achieve this goal, the spike time t_j is delayed to $\tilde{t_j}$ by the function $f(u_j)$, as shown in Fig. 3.1b, where the function $f(\cdot)$ represents a non-linear activation of the selected dendritic segment u_j, with j being the index of the post-synaptic neuron. It is important to note that, differently from Iyer et al., the dendritic branch of our proposed solution is a vector $u \in \mathbb{R}^n$, rather than a matrix $d \in \mathbb{R}^{n \times n}$, with n being the number

of tasks. In other words, in our model, each dendritic segment contains a single parameter, while in Iyer et al. each segment contains a vector of n parameters. The parameter u_j related the current task is extracted as

$$u_j = u \cdot c, \tag{3.1}$$

where c is a one-hot-encoded vector with dimension $\mathbb{R}^n$ representing the current task at play. This operation implements a simple task-dependent switching mechanism that selects different dendritic segments for different tasks.



(a) Proposed neuron model **(b)** spike Time Delay

Figure 3.1: Proposed solution. **(a)** Rel-PSP model enhanced with *active dendrites*. It is composed of n dendritic segments, each representative of a different task and each possessing a training parameter u. Depending on the one-hot encoded vector c, the respective dendritic branch parameter u_j passes through the non-linear function $f(\cdot)$, thereby introducing a delay in the spike time. In **(b)** the integration of the membrane potential is depicted. As one can see, the actual spike time t_j^l is shifted to $\tilde{t}_j^l$ due to the effect of the dendritic delay introduced by $f(u_j)$. This behavior effectively modulates neuron activity similar to the solution proposed by Iyer et al..

The selected dendritic segment u_j is passed through the activation function $f(\cdot)$ and used to modulate the spike time of the neuron. To effectively introduce the modulation delay $f(u_j)$, the kernel is defined as

$$\epsilon[t] = \begin{cases} t - t_i - f(u_j), & if\ t > t_i + f(u_j) \\ 0, & otherwise. \end{cases} \tag{3.2}$$

Consequently, the evolution of the membrane potential of a single neuron becomes

$$V_m[t] = \begin{cases} \sum_i^I W_{ij}[t - t_i - f(u_j)], & if\ t > t_i + f(u_j) \\ 0, & otherwise. \end{cases} \tag{3.3}$$

Differently from the original model formulation of Eq. (2.5), where the membrane potential integration starts when a pre-synaptic neuron emits a spike, in Eq. (3.3), the membrane potential integration is delayed by a factor $f(u_j)$ following the event of a pre-synaptic spike. Consequently, the effective spike time of the proposed neuron model is exactly delayed by the function $f(u_j)$ as defined in Eq. (3.4).

$$\tilde{t_j} = \frac{V_{th} + \sum_i^N W_{ij} t_i}{\sum_i^N W_{ij}} + f(u_j). \tag{3.4}$$

The dendritic activation function, i.e. $f(\cdot)$, plays a fundamental role in modeling the amount of delay introduced in the spike time. This activation function is defined as

$$f(u_j) = N^{\frac{1}{1+e^{u_j}}} - 1. \tag{3.5}$$

In Eq. (3.5), the parameter N is a hyperparameter that can be changed to increase the modulation effect of active dendrites. This function has been designed so that when the parameter u_j increases towards positive values, the delay is reduced, and when it decreases towards negative values, the delay increases, as shown in Fig. 3.2. Moreover, this function is differentiable with respect to parameter u_j, thus allowing the gradients to be calculated as per the back-propagation algorithm.

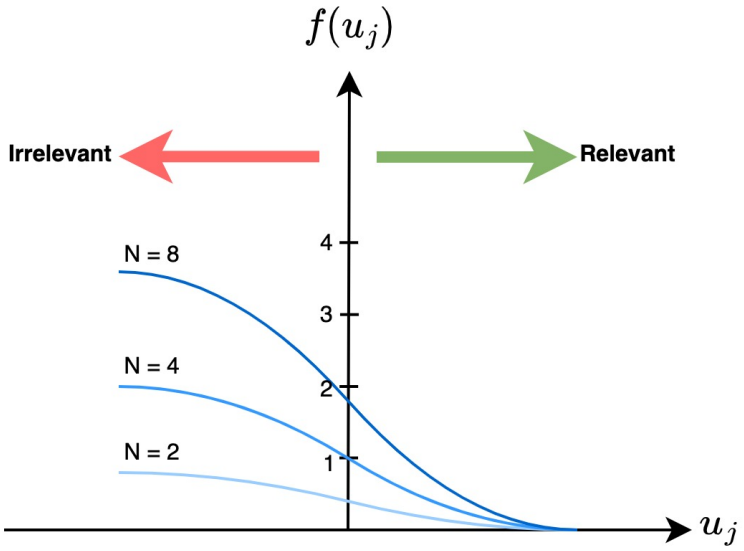


Figure 3.2: Dendrites activation function. The activation function is depicted for different values of N. At the beginning of training, the parameter u_j is initialized to 0. Depending on the direction of the gradient update, u_j can move either to positive or negative values, respectively decreasing or increasing the dendritic delay $f(u_j)$. If the neuron is relevant to the task, the delay is reduced, otherwise, the delay is increased. This mechanism ensures a modulation of the neuron's activity depending from the relevance to the current task.

It is worth highlighting that there is a major conceptual difference between the model proposed by Iyer et al. and ours. For instance, to ensure that each dendritic segment can learn to recognize the context vector of each task, Iyer et al. assume that there are n dendritic segments, each containing a vector of n parameters, resulting in a total of n^2 parameters for the active dendrites of a single neuron. By following this approach and applying the selection mechanism of Eq. (2.19), Iyer et al. ensure that the network can incorporate new tasks without having to add new dendritic segments. However, the authors have shown that not all dendritic segments are required to learn a small number of tasks. As a consequence, to reduce the complexity in light of a

hardware implementation, our model includes a single parameter for each dendritic segment, resulting in a total of n parameters for a single neuron.

3.1.2. Network architecture

As mentioned in the previous chapter, gating specific neurons in a task-dependent manner plays a fundamental role in mitigating the problem of catastrophic forgetting. To achieve this task-dependent gating, Iyer et al. add a *k*-WTA layer after the hidden layer enhanced with active dendrites. The *k*-WTA layer is essential in artificial neural networks to set the activation values of the *k* least active neurons to zero, thus removing their gradients in the backward pass. On the contrary, we hypothesize that the issue of dead neurons inherent in a TTFS spiking network behaves similarly to a *k*-WTA layer. The architecture used in our software simulation is depicted Fig. 3.3 for two different tasks.

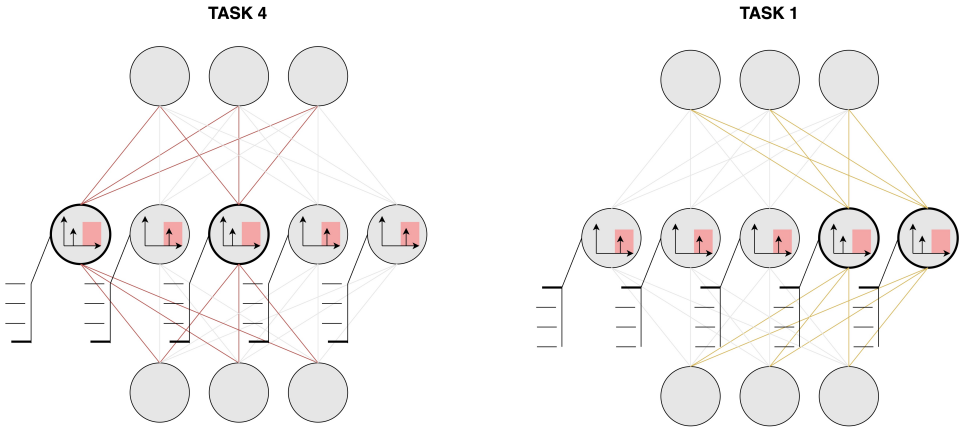


Figure 3.3: Architecture and sub-networks of a Rel-PSP model enhanced with *active dendrites*. The same network architecture is depicted for two different tasks, e.g. task 4 and task 1. The brush-like structures connected to each hidden neuron are representative of the active dendrites. The thickest bristle represents the dendritic segment of the current active task. Inside each neuron, a small temporal plot represents the spike time, illustrating when the neuron fires. The red area represents the time range after which the neuron is *dead*. In the hidden layer, neurons with thick black lines are active, while the others are gated. Note that different tasks activate distinct neurons based on the connected dendritic segment parameter, thereby creating different subnetworks for each task and reducing information interference during training.

A neuron is considered *dead* if its spike time occurs after the observation window T_{max}, making it impossible to calculate its gradients. Moreover, a dead neuron is equivalent to a real value of zero in a network of artificial neurons, thereby acting as a gating mechanism similar to the one employed by the *k*-WTA layer. Since our proposed neural model introduces a task-dependent delay in the spike time of a neuron, its probability of being dead changes depending on the current task. For instance, if a neuron is irrelevant for a given task, its delay will be increased by the back-propagation algorithm, thereby increasing the chance of being dead in the next iteration. By following this approach, different neurons in the hidden layer are gated based on the current task. For each of these neurons, only those that are relevant to the current task are not gated and thus updated, thereby reducing the interference of information between different tasks.

3.1.3. Training

In Section 2.4.3, an SRM with a Rel-PSP kernel is trained with the conventional back-propagation algorithm. However, to apply this algorithm to the proposed model, its computational graph must be modified to account for the additional computations required to incorporate the active dendrites. The new computational graph, which encompasses both forward and backward propagation, is illustrated in Fig. 3.4. As shown in this figure, the only additional computational path, compared to the standard Rel-PSP kernel, is the one defined by the dendritic parameter u_j.

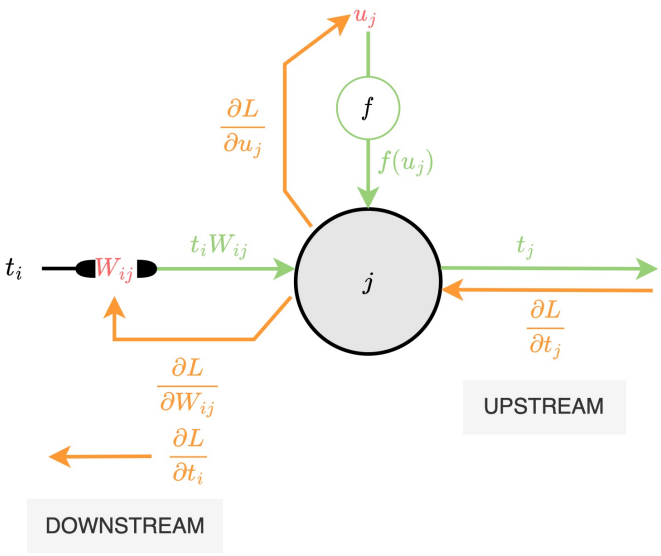


Figure 3.4: Computational graph of the proposed solution. The green lines represent the forward propagation while the orange ones the backward propagation. The trainable parameters are colored in red, i.e. W_{ij} and u_j.

The downstream gradient $\partial L/\partial t_j^l$ comes from the next layer and depends on the loss function expressed in Equation 2.17. Following the introduction of the modulation effect, the gradient required to update the neuronal parameters W_{ij} changes to

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial t_j}{\partial V[t_j]} \frac{\partial V[t_j]}{\partial W_{ij}} \frac{\partial L}{\partial t_j} = \frac{-t_j + t_i + f(u_j)}{\sum_i^I W_{ij}} \frac{\partial L}{\partial t_j}. \tag{3.6}$$

On the other hand, the gradient required to update the dendritic parameter u_j is

$$\frac{\partial L}{\partial u_j} = \frac{\partial t_j}{\partial V[t_j]} \frac{\partial V[t_j]}{\partial u_j} \frac{\partial L}{\partial t_j} = \frac{f'(u_j)}{\sum_i^I W_{ij}} \frac{\partial L}{\partial t_j}, \tag{3.7}$$

where $f'(u_j)$ is the derivative of the activation function $f(\cdot)$ found as

$$f(u_j)' = \frac{N^{\frac{1}{e^{u_j}+1}} ln(N) e^{u_j}}{(1+e^{u_j})^2}. \tag{3.8}$$

The downstream gradient, i.e. $\partial L/\partial t_i$, is not affected by the introduction of the active dendrites and remains the same as defined in Eq. (2.16). Both the forward and backward paths have been implemented using the PyTorch neural network framework.

3.2. Hardware design

This section presents a novel hardware implementation incorporating active dendrites to solve the problem of catastrophic forgetting in a TTFS-encoded neural network. The section starts by presenting the hardware architecture, followed by an explanation of the neuron processing unit (NPU), the neural cluster, the memory organization, and the controllers. The proposed solution has been implemented on a Xilinx Zynq-7020 SoC FPGA using the *Verilog* hardware description language. To ensure the matching between the software implementation and the hardware implementation, the state of each neuron is updated at each simulation time step[1]. The detailed Verilog code implementation can be found here.

3.2.1. Architecture

The power efficiency of parallel architectures can be improved if properly designed for the encoding mechanism of choice. For example, in TTFS-encoded neural networks, after a neuron in a layer has spiked, it does not need to process any additional incoming spikes. Since the dynamic power of a digital system is directly proportional to the *switching activity*, which represents the rate of transitions between logic states in the system, TTFS encoding can reduce this activity by minimizing unnecessary transitions and conserving power resources. In essence, a physical neuron within the cluster of a layer can be selectively disabled after a spike, thereby reducing dynamic power consumption. To exploit these capabilities, a parallel architecture similar to the one explained in Section 2.6.1 is utilized. The architecture of a single layer is shown in Fig. 3.5.
The proposed layer architecture consists of the following blocks:

- **Main controller:** finite-state machine (FSM) managing the processing controller, the memory controller, the neural cluster, and the output queue. It implements a 4-phase handshake protocol to communicate with other layers. A detailed description of this block is provided in Section 3.2.4.
- **Processing controller**: FSM that processes the spikes in the input queue. A detailed description of this block is provided in Section 3.2.4.
- **Input queue:** first-in-first-out (FIFO) queue storing the memory addresses of the incoming spikes from the previous layer.
- **Output queue:** FIFO queue storing the memory address of the neurons that generated a spike in the current layer. Addresses within this queue are pushed to the input queue of the next layer when layer processing is complete.
- **Memory controller**: FSM controlling memory accesses to the synapse and active dendrites memories. A detailed description of this block is provided in Section 3.2.4.
- **Synapse memory:** block-random-access memory (BRAM) containing synaptic strength parameters connecting the neurons of one layer with the previous. The memory organization of this block is provided in Section 3.2.3.

[1] Discrete unit of time used to process pre-synaptic spikes and update post-synaptic neurons.

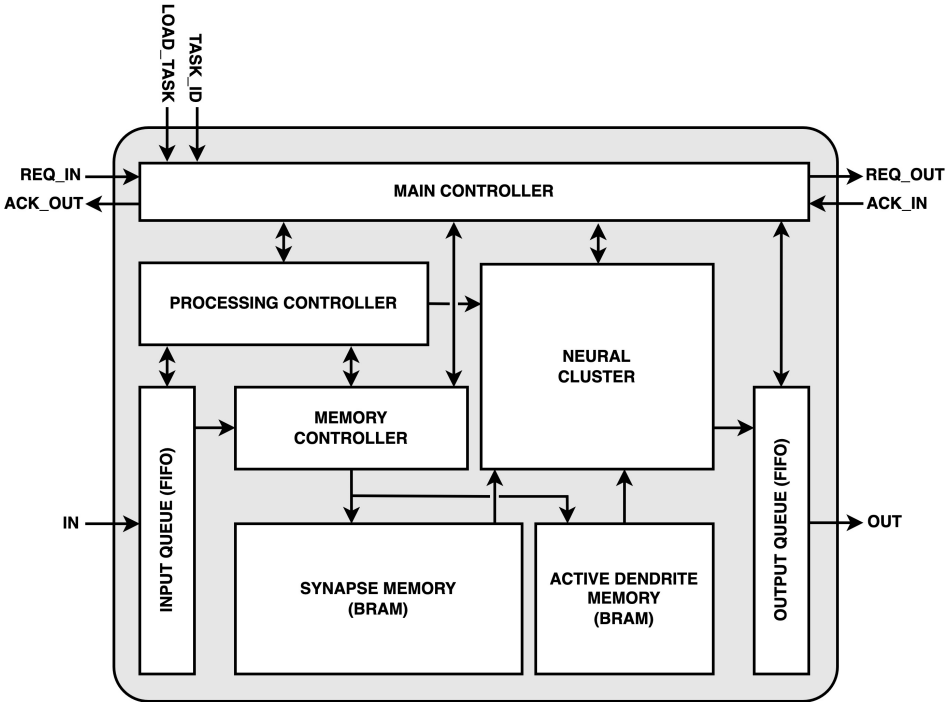


Figure 3.5: Proposed hardware layer. Hardware architecture of the proposed solution showing the different components of a given layer. The arrows represents the data flow of information.

- **Active dendrites memory:** BRAM containing dendritic delays, i.e. $f(u_j)$ for each neuron in the layer. The memory organization of this block is provided in Section 3.2.3.
- **Neural cluster:** array of neuron processing unit (NPU), described in Section 3.2.2. It receives synaptic parameters and delays from BRAM and updates the internal states of each neuron. A detailed description of this block is provided in Section 3.2.3.

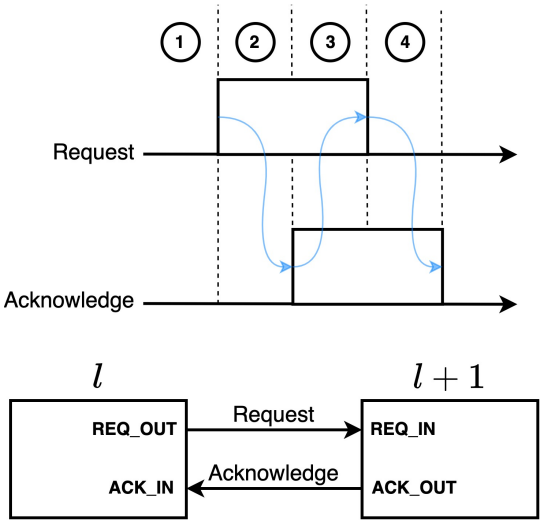


Figure 3.6: 4-phase handshake protocol. The signal transition between two consecutive layers takes place in four different different phases, shown in numbered circles.

The communication of spikes between layers is managed by a 4-phase handshake protocol, as depicted in Fig. 3.6. During Phase 1, the input spikes that occur within a simulation time step t_{step}, in a given layer l, are accumulated in the input spike queue of the next layer $l+1$. Following this accumulation, layer l, sends a request signal to layer $l+1$, which initiates Phase 2. During this phase, the states, e.g. membrane potentials, of neurons in the neural cluster of layer $l+1$ are updated and compared against the threshold voltage. If a spike is emitted, the address of the spiking neuron is stored in the output queue of layer $l+1$.

Following this processing phase, layer $l+1$ generates an acknowledgment signal, thus initiating Phase 3 of the handshake protocol. During this phase, the FSMs of the controllers in layer l are reset to their idle states. Subsequently, layer l sets the request to zero, thus initiating Phase 4. During this phase, all the consecutive layers process their input queues following the same scheme. At completion, the acknowledgment bit is set to zero, signaling the completion of processing of the current simulation time step, from all layers.

3.2.2. Neuron processing unit (NPU)

The neuron processing unit (NPU) implements the neural dynamics of an SRM model with a Rel-PSP kernel, enhanced with active dendrites as defined in Section 3.1.1. The digital implementation of the proposed NPU is depicted in Fig. 3.7.

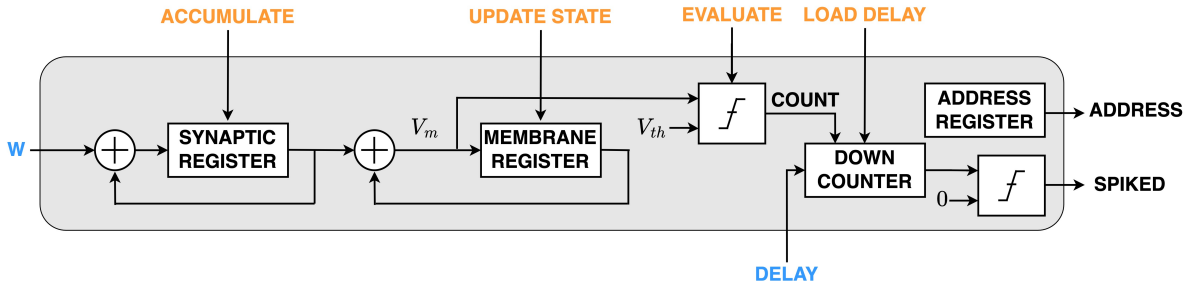


Figure 3.7: Neuron processing unit (NPU). Hardware implementation of the proposed neuron model. Signals in orange are from the main controller while signals in blue are from the BRAM. The down counter contains the delay $f(u_j)$ introduced by the active dendrites.

Within a particular simulation time step, a single post-synaptic neuron in layer l might receive spikes from different pre-synaptic neurons in layer $l-1$. The addresses of the pre-synaptic neurons that emit a spike in layer $l-1$ are stored in the input queue of layer l. In accordance with Eq. (3.3), the synaptic strength parameters W_{ij} connecting the spiking pre-synaptic neurons to the post-synaptic neuron are accumulated.

To implement the accumulation mechanism, when a pre-synaptic spike is retrieved from the input queue, the memory controller retrieves the corresponding synaptic parameter from the BRAM and redirects it to the NPU input, represented by the blue W in Fig. 3.7. Subsequently, the main controller sends an accumulation signal (accumulate in Fig. 3.7), which stores the parameter in the synaptic register. When a new spike is retrieved from the queue, the same process is performed, buffering the sum of the parameters of the spiking pre-synaptic neurons in the synaptic register. This process is performed until the input queue is empty.

As the simulation progresses, the accumulated parameters are integrated in the membrane register when the update state signal transitions to a high state, thus implementing the complete discrete-time evolution of the membrane potential of Eq. (3.3). At the end of each simulation time step, the membrane voltage is compared with the threshold voltage V_{th}. If its value exceeds the threshold, the count signal transitions to a high state. This event corresponds to the occurrence of a spike in a standard SRM model with a Rel-PSP kernel without active dendrites.

As explained in Section 3.1.1, the active dendrites of a neuron introduce a delay in the time it takes for the neuron to spike. This delay can be modeled by a counter, called the down counter. The down counter is initialized with the delay of the given task, i.e. $f(u_j)$, contained in the BRAM. When the count signal transitions to a high state, the down counter starts decrementing by one with each new simulation step until it reaches zero. At this point, i.e. $t_{sim} = \tilde{t}_j$, the neuron emits a spike by raising the spike signal to a high state. An example timing diagram of these transitions is shown in Fig. 3.8.

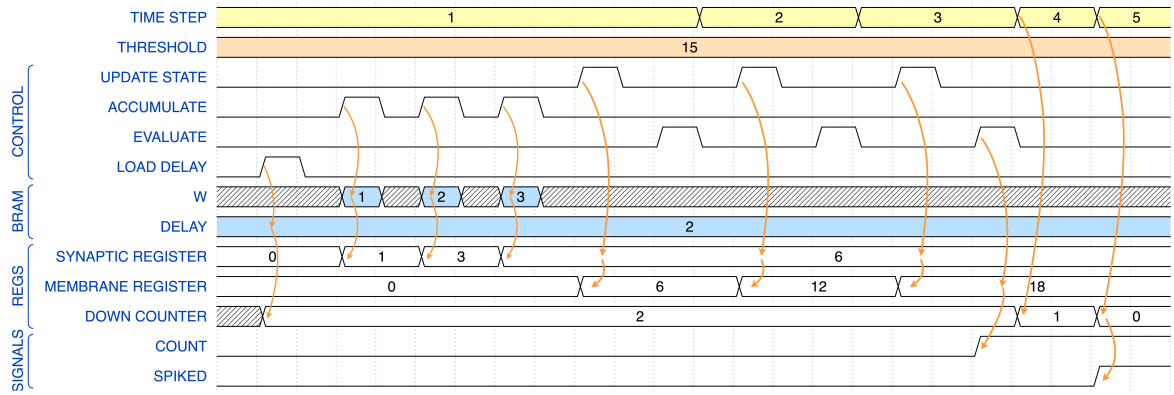
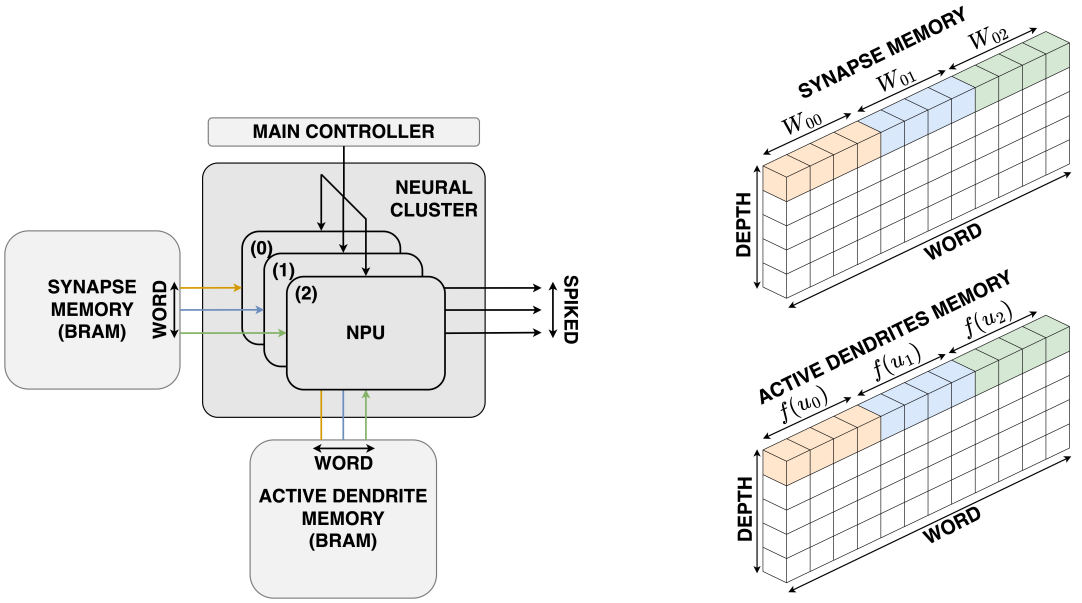


Figure 3.8: Timing diagram of a NPU. At the beginning of the first time step, the DELAY is loaded into the down counter, following the transition to a high state of the LOAD DELAY signal. In this example, during the first simulation step, the post-synaptic neuron receives 3 spikes from different pre-synaptic neurons. These spikes are decoded and the relative synaptic parameters, i.e. $W = 1$, $W = 2$, and $W = 3$, are directed to the input of the NPU from the BRAM. In this specific example, the post-synaptic neuron does not receive any new spike in the following time steps. When the value in the membrane register crosses the threshold, i.e. $V_m > V_{th}$, the count signal transitions to a high state, thus starting the down counter. When this counter value reaches zero, the spiked signal transitions to a high state, indicating the event of a spike.

3.2.3. Memory organization and neural cluster

Each layer contains two physical memories: the synapse memory and the active dendrite memory. These memories are implemented using the BRAM available in the programmable logic (PL) of the Zynq-7020 SoC. Specifically, the Zynq-7020 has a total memory of 8.4 kB organized in 140 blocks of 4.8 kb each. The memories in each layer are instantiated using the Xilinx Block Memory Generator IP. This tool enables the user to specify the depth and word length of each memory. The tool then arranges the different blocks so that the required memory appears as a single unified memory block to the user.

As discussed in Section 2.4.1, an SNN consists of multiple layers of neurons, each



(a) Neural cluster **(b)** Memory organization

Figure 3.9: Neural cluster and memory organization. **(a)** Neural cluster containing three neural processing units, i.e. units (0), (1) and (2). Each NPU is connected to the synapse and active dendrites memories. The connections to the main controller are shared between the NPUs. **(b)** Memory organization of the synapse memory (top) and the active dendrites memory (bottom) is shown for three post-synaptic neurons. The colors indicate the connections between the memories and the cluster. For example, W_{00} and $f(u_0)$, represented in orange, are directly connected to unit (0).

layer connected to the next by a synaptic connection matrix $W^l \in \mathbb{R}^{I \times J}$, where J is the number of post-synaptic neurons in layer l, and I is the number of pre-synaptic neurons layer $l-1$. The matrix entries W_{ij} are stored in the BRAM of the synapse memory with a bit precision of N_W.

To implement this layered structure, the neural cluster in a given layer l contains multiple instances of NPUs implementing the dynamics of neurons in that layer. Signals from the main controller are shared between units, allowing their updates to occur in parallel. In addition, each NPU is directly connected to the output ports of the synapse and active dendrite BRAMs. This connectivity ensures direct access to synaptic parameters and active dendritic delay values, allowing the NPU to receive the parameters required for integration in parallel.

Consider, for example, the three post-synaptic NPUs depicted in Fig. 3.9a, specifically units (0), (1) and (2). According to Eq. (3.3), which describes the evolution of the membrane potential of the proposed model, if the pre-synaptic neuron $i = 0$ emits a spike in a given simulation step, unit (0) requires synaptic parameter W_{00}, unit (1) requires W_{01} and unit (2) requires W_{02} to update their states. To receive these parameters in parallel, the synapse memory is organized as depicted in Fig. 3.9b. The depth of the synapse memory is determined by the number of neurons I in the pre-synaptic layer, while the word length is given by the product of the number of post-synaptic neurons and the precision of each parameter, i.e. $J \times N_W$.

According to this memory organization, when the address of a spiking pre-synaptic neurons is retrieved from the input queue of the post-synaptic layer, the memory con-

troller selects the corresponding location in the synapse memory, thereby connecting each parameter to the respective NPU. Following our initial example, since the pre-synaptic neuron $i = 0$ emitted a spike, the memory controller selects the address 0 from the synapse memory, thus connecting W_{00} to unit (0), W_{01} to unit (1) and W_{02} to unit (2). Subsequently, the main controller sends the control signals explained in Section 3.2.2, to update the states of the NPUs.

Least but not last, to implement the delay effect of active dendrites, the down counter of each NPU must be initialized with the delay of the current task. Delay values are stored in the active dendrite BRAM with precision N_u. To achieve parallel initialization of the down counters, the active dendrite memory is organized as shown in the bottom of Fig. 3.9b. Specifically, memory depth is determined by the number of dendritic segments which, as explained in Section 3.1.1, is equal to the number of tasks, i.e. n. On the other hand, the length of the word is determined by the product of the number of post-synaptic neurons and the precision of each delay, i.e. $J \times N_u$.

Using the same example as in Fig. 3.9, if the first task is being performed, the delay values of the first dendritic segment of each neuron, i.e. $f(u_0)$, $f(u_1)$ and $f(u_2)$, must be loaded onto the down counter of the respective neurons, i.e. units (0), (1) and (2). To achieve this goal, the memory controller selects the first address of the active dendrite BRAM, connecting the delay $f(u_0)$ to unit (0), $f(u_1)$ to unit (1), and $f(u_2)$ to unit (2). Subsequently, the main controller raises the load delay signal to a high state, thereby loading all the delays in the respective down counters.

3.2.4. Controllers

The proposed layer architecture contains three controllers: the main controller, the processing controller, and the memory controller implemented as a hierarchical finite-state machine, as shown in Fig. 3.10.

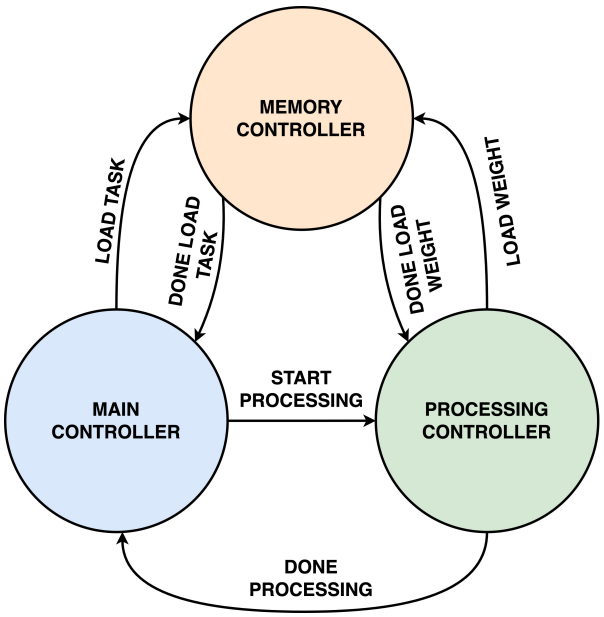


Figure 3.10: Hierarchical FSM. Interactions among the main controller (orange), the processing controller (green), and the memory controller (blue).

The main controller is the fundamental block of a layer. It implements the 4-phase handshake protocol to communicate with adjacent layers. It updates the membrane potential of each NPU. It evaluates the occurrence of a spike, and subsequently adds the addresses of any spiking neurons to the output queue. The finite state machine of this controller is depicted in Fig. 3.11a. It consists of the following states:

- **IDLE**: spikes in a given simulation step from the previous layer are stored in the input queue,
- **LOAD TASK**: sends a signal to start the memory controller with the aim of loading the active dendrites delays,
- **START PROCESSING**: sends the signal to start the processing controller,
- **EVALUATE**: checks the membrane potential of each NPU in the neural cluster against the threshold voltage, pushing the address of any NPU that crosses the threshold into the output queue,
- **UPDATE POTENTIAL**: updates the membrane potential of each NPU based on the accumulated spikes from the processing controller,
- **POP OUT**: pops an element from the output queue,
- **PUSH OUT**: pushes the popped elements into the input queue of the next layer,
- **DONE**: the current simulation step of the layer is completed, and the controller remains in this state until it receives an acknowledgment signal from the next layer.

The purpose of the processing controller is to process the spikes accumulated in the input queue from the previous layer. Specifically, it pops elements from the input queue until it is empty and accumulates the respective synaptic parameters in the synapse register of each NPU. The FSM of this controller is depicted in Fig. 3.11b. It consists of the following states:

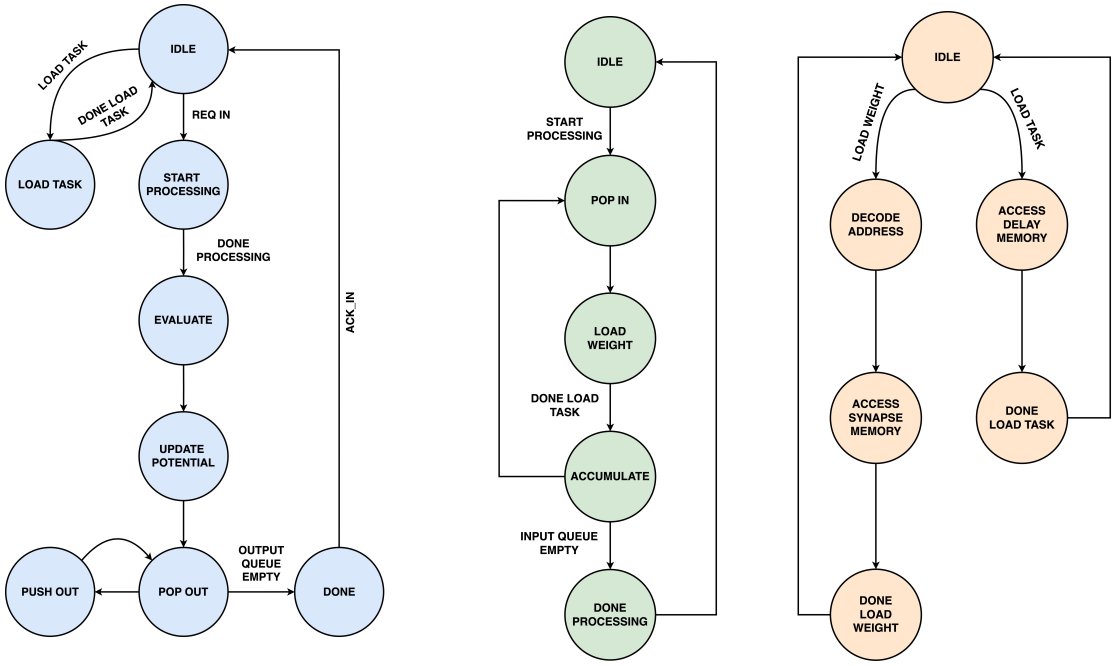
- **IDLE**: waits for the start signals from the main controller,
- **POP IN**: pops one element from the input queue,
- **LOAD WEIGHT**: sends a signal to activate the memory controller, which will decode the popped element and access the synapse memory,
- **ACCUMULATE**: sends and accumulate signal to each of the NPUs in the neural cluster and checks if the input queue is empty,
- **DONE PROCESSING**: sends the done processing signal to the main controller, forcing a transition to the next state.

The memory controller manages accesses to the synapse and active dendrite BRAMs. The finite state machine of this controller is depicted in Fig. 3.11c. It consists of the following states:

- **ACCESS DELAY MEMORY**: sends a read signal to the active dendrite memory,
- **DONE LOAD TASK**: sends a done signal to return to the idle state of the main controller,

- **DECODE ADDRESS**: decodes the address in the input queue to a memory address of the synapse memory,[2]
- **ACCESS SYNAPSE MEMORY**: sends a read signal to the synapse memory,
- **DONE LOAD WEIGHT**: sends a done signal to return to the idle state of the main controller.

It should be highlighted that the BRAM in the Zynq-7020 SoC requires two clock cycles to access the data at a given address with the output register enabled. As a consequence, each time the memory controller enters one of the two memory access states, it remains in that state for two clock cycles while keeping the read signal to a high state.



(a) Main controller **(b)** Processing controller **(c)** Memory controller

Figure 3.11: Controllers states. The state machine for each controller is displayed following the same color convention of Fig. 3.10. For the purpose of clarity, *self-loops* of states are not depicted. The name of the signal adjacent to each arrow connecting two states represents the transition signal. If this signal is not received, the state does not change. Arrows connecting states without specified transition signals denote transitions independent of other signals.

[2]This state is empty in our experimental implementation because the synaptic parameters connecting a specific pre-synaptic neuron to all the post-synaptic neurons can be written in a single word of memory. Hence, there is a one-to-one mapping between the address of the pre-synaptic neuron and the memory address. If the number of post-synaptic neurons is greater than $4608/N_{bits}$, a decoding mechanism is required.

4

Results

This chapter provides the results obtained with the proposed solutions, encompassing both the software and hardware perspectives. First, we introduce the reader to the case study used to evaluate the ability of our solution to mitigate the problem of catastrophic forgetting. Second, we present software simulation results for two models: one enhanced with active dendrites and another without, highlighting the model's performance in terms of memory retention within a continual learning environment. Finally, we present the hardware results of the proposed architecture, highlighting its efficacy in terms of latency, resource utilization, and power.

4.1. Case study

In the field of neuromorphic computing, the Neuromorphic-MNIST (N-MNIST) dataset [14] is a common dataset used to benchmark a model. This dataset is a spiking version of the conventional frame-based MNIST dataset [14] employed by the machine learning community.

The N-MNIST dataset is realized by recording the output of an event-based ATIS dynamic vision sensor (DVS) [64] mounted on a motor-controlled pan-tilt unit. The sensor is directed towards a monitor displaying all the images of the frame-based MNIST dataset. To generate output spikes, the DVS is moved in three different directions, to mimic three saccades of a human eye. An example of the output of the DVS camera while recording a zero digit is shown in Fig. 4.1. The recorded output from a DVS camera is a list of events, where each event is represented by 40 bits as described below:

- bits 0-22: Timestamp (μs)
- bits 23: Polarity (increase or decrease in light intensity)
- bits 24-31: Y address (pixels)
- bits 32-39: X address (pixels)

However, conventional neural networks can only process tensor formats and cannot effectively handle a list of events. The temporal resolution of the ATIS camera is on the order of microseconds and the total average duration of each sample image is

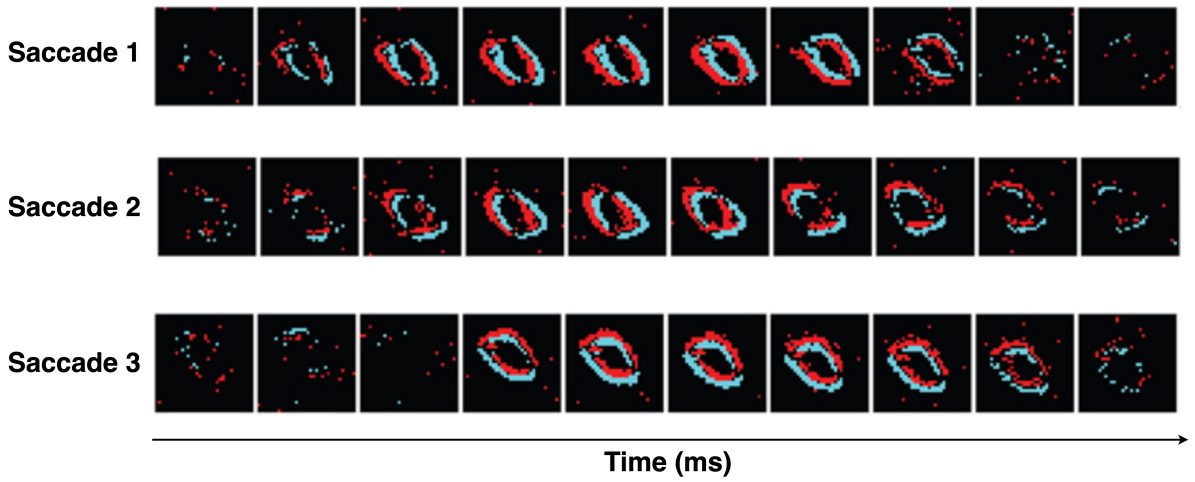


Figure 4.1: N-MNIST saccades. Output of the DVS camera for three saccades while viewing digit zero of the frame-based MNIST dataset. Image modified from [14]. Red spikes represent an increase in light intensity, whereas blue ones represent a decrease in light intensity.

of 100 ms for each saccade. Thus, attempting to represent these events in a tensor format would result in a temporal dimension of the order of 10^5 steps, which makes it intractable in terms of training time. Consequently, to reduce training time, events are conventionally binned into frames [65]. Specifically, in our experimental setup, events that occur within a time window of $1\ ms$ are binned together.

Furthermore, as Frenkel et al. demonstrated in [58], using only the first saccade is sufficient to achieve good levels of performance in classification tasks. Consequently, to further reduce training time without sacrificing performance, we only consider only the spikes that occur within the first saccade.

Finally, also similar to [58], each image in the dataset needs to be converted to a TTFS encoding. Hence, only the first spike event for each pixel is retained, while all the following spikes of the same pixel are removed. Lastly, only the events of increasing light intensity have been considered for all our experiments, i.e. only one channel is used.

4.2. Software results

This section provides the software results of the proposed solution. It is divided into two subsections. The first provides quantitative results of the proposed model, i.e. an SRM with Rel-PSP kernel enhanced with active dendrites, in typical continual learning environment, compared to the same model without active dendrites. The second provides a discussion of these results. Specifically, it focuses on discussing the emergence of subnetworks for each task and the specialization of each neuron to each task.

4.2.1. Continual learning results

Indeed, as previously explained in Section 2.5.2, a typical neuromorphic experiment to benchmark continual learning solutions consists in sequentially feeding images of digits to the network during the training phase. In other words, tasks, each containing

all images of a single digit, are presented sequentially to the network.

To create this environmental setup, the N-MNIST dataset has been modified to present the digits to the model in a sequential order, with all samples of one digit at a time. N-MNIST contains 60000 training samples and 10000 testing samples. However, the number of samples for each digit is not balanced. Consequently, to ensure a balanced number of samples for each task, the dataset has been modified to include 5000 training samples and 890 test samples for each digit.

To evaluate the performance of the proposed model, we use the average accuracy metric across 5 tasks, i.e. digits 0, 1, 2, 3, and 5, at the end of the training session as described in Section 2.5.1. For each experiment, we maintain two models: one enhanced with active dendrites and one without, where the former is expected to alleviate the catastrophic forgetting problem of the latter.

The images of a specific task are grouped into batches of 256 elements. Each task is trained for 5 epochs using the *Adam* optimizer. To ensure statistical validity, the experimental results are averaged across 4 different seeds. The neural network employed for all our experiments is a one-hidden-layer model. The input layer contains 1156 neurons, one for each pixel of the DVS camera, the output layer contains 5 neurons, one for each digit. Finally, the hidden layer contains 256 neurons. The synaptic connections of each model are initialized using a He [66] distribution.

To find the best-performing model, we performed a grid search across various hyperparameters such as the active dendrites strength N of Eq. (3.5), the learning rate of the dendrites η_u, and the learning rate of the synaptic weights η_W of Eq. (2.10). The results of this experiment are summarized in Table 4.1.

N	η_W	$\eta_u = 0.001$		$\eta_u = 0.01$		$\eta_u = 0.1$	
3	0.00001	0.23	0.45	0.23	0.8	0.23	0.85
	0.0001	0.38	0.4	0.38	0.52	0.38	0.67
	0.001	0.23	0.25	0.23	0.25	0.23	0.31
30	0.00001	0.23	0.98	0.23	1.00	0.23	1.00
	0.0001	0.38	0.83	0.38	1.00	0.38	1.00
	0.001	0.23	0.50	0.23	0.64	0.23	0.75
300	0.00001	0.23	0.20	0.23	0.23	0.23	0.2
	0.0001	0.20	0.38	0.20	0.38	0.20	0.38
	0.001	0.23	0.20	0.23	0.20	0.23	0.20

Table 4.1: Average test accuracy expressed in decimal values across 5 tasks measured at the end of training, i.e. all digits have been presented to the network. Each entry contains two sets of results. On a dark gray background, are the test accuracy results for the model enhanced active dendrites, while on a light gray background are the test accuracy results for the model without active dendrites.

To gain deeper insights into the effectiveness of our proposed solution in mitigating the problem of catastrophic forgetting, it is possible to generate plots such as the one depicted in Fig. 4.2. These plots are generated by evaluating the network's test accuracy at each training step, considering all previous tasks and training epochs.

For the specific model used to generate the plots shown in Fig. 4.2, we set the network's hyperparameters as follows: $N = 30$, $\eta_W = 0.0001$ and $\eta_u = 0.1$. Notably, in the model without active dendrites, depicted on the left side of Fig. 4.2, the accuracy on the test set of previously-learned tasks decreases as new tasks are encountered.

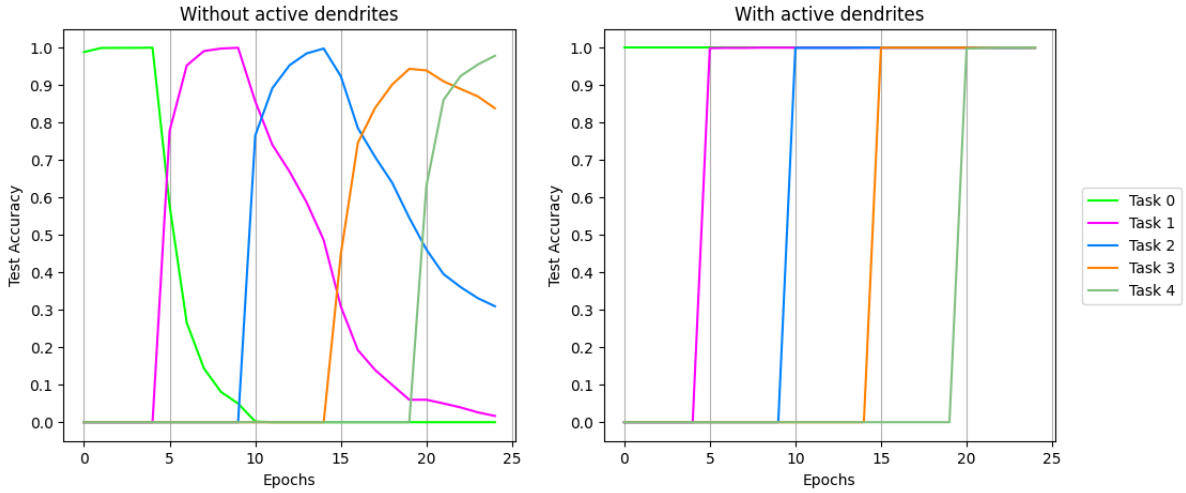


Figure 4.2: Test accuracy at each epoch. On the left side, the test accuracy at each step for the model without active dendrites is shown. On the right side, the test accuracy at each step for the model with active dendrites is. The beginning of training for each new task is marked by the vertical lines every 5 epochs. The different colors represent the different task. For example, the green color represents task 0, where only images of the digit zero are presented to the network.

For instance, consider the fifth epoch in the figure on the left, during which the second task performs its first training step. As one can see, during this step, the accuracy of the first task, depicted in green, drops from 100% to around 60%, highlighting the problem of catastrophic forgetting. On the contrary, the model enhanced with active dendrites maintains the same test accuracy when the first iteration of the next task begins.

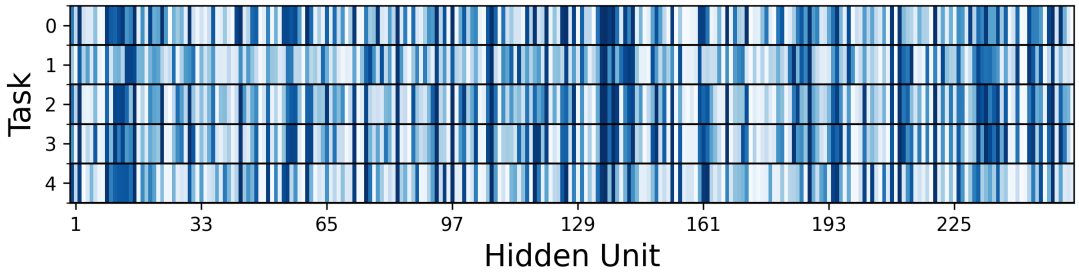
4.2.2. Discussion

The results provided in Table 4.1 suggest that the proposed solution is capable of mitigating the catastrophic forgetting problem. However, the results shown in the right plot of Fig. 4.2 raise many questions. Indeed, it is highly unlikely to obtain a final average accuracy of 100% in all tasks. There are multiple reasons that could cause this event to occur. In the best-case scenario, this high accuracy can be attributed to the fact that the tasks are too simple and that each task is trained by a different subnetworks, therefore removing the interference of other digits. In the worst-case scenario, this high accuracy could be attributed to repetition of digits within the same task. The latter has been experimentally tested, and there is no identical image within the same task. It is worth highlighting that other works attempting to mitigate catastrophic forgetting under the same continual learning setup, also archive very high accuracy. For example, the best performing model in [40] achieves an average accuracy of 99% at the end of the training phase of the first five digits. Notably, similar to our results, in [40, 38] the average accuracy for the first two digits is also 100%.

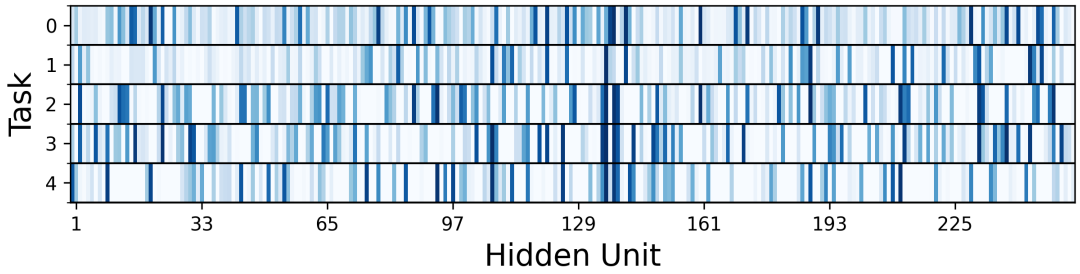
Furthermore, when the strength of the active dendrites is large, e.g. $N = 300$, the proposed solution performs worse than the model without active dendrites. This is mainly due to the design of the activation function of the dendritic segment defined in Eq. (3.5). If the activation strength is too large, i.e. N is large, each dendrite introduces

a large delay in the spike time, resulting in a large number of dead neurons, which in turn hinders the possibility of calculating error gradients. On the other hand, if the activation strength is reduced, a smaller number of neurons become dead, allowing for a better calculation of error gradients. As explained in Section 3.1.2, we hypothesized that dead neurons act like a task-dependent gating mechanism which ensure that only the neurons relevant to a specific task remain active during the forward propagation. This gating mechanism generates different subnetworks for different tasks, thereby reducing the interference of information between tasks.

To test this hypothesis, we measured the activity of each neuron across tasks, where the neuron activity is defined as the number of spikes emitted during the presentation of images associated with a specific task. This measurement enables us to gain insight into the neural response in relation to each task. A plot of the activity of each neuron in the hidden layer with respect to each task is depicted in Fig. 4.3. In this figure, a dark blue line indicates a large number of spikes, whereas a light blue line represents a small number of spikes.



(a) Activity without active dendrites



(b) Activity with active dendrites

Figure 4.3: Neurons activity for different tasks. The horizontal axis represents the neuron, whereas the vertical represents the current task. The color represents the neuron activity, the darker is the color the more active is the neuron.

From this figure, it is evident that active dendrites increase sparsity in the neural activity of the hidden layer, thereby validating our hypothesis that dead neurons act as a gating mechanism. Furthermore, when the model is enhanced with active dendrites, the activity of each neurons becomes more specific to a given subset of tasks.

To gain a better understanding of the degree to which each neuron is responsive to each task, we define a neuron as being highly responsive to a given task if it generates an output spike for at least half of the image samples of that task. By employing this metric, we generated the graph shown in Fig. 4.4, where the orange columns represent the model without active dendrites, while the blue columns represent the

model with active dendrites. From this figure, it is evident that when the model is enhanced with active dendrites, each neuron becomes more responsive to a subset of the total tasks.

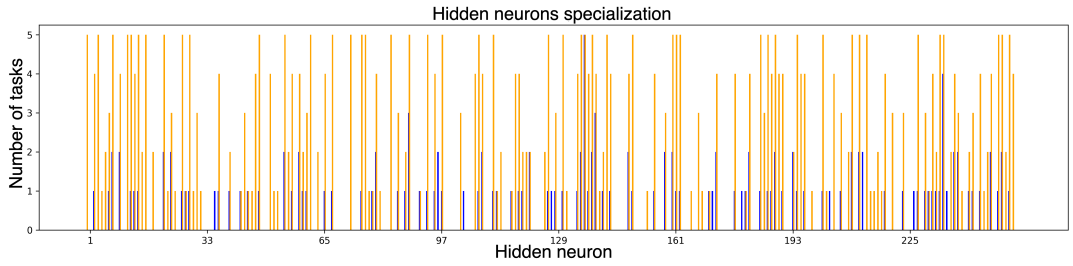


Figure 4.4: Neurons specialization. The horizontal axis represents the neurons in the hidden layer. The vertical axis represents the number of tasks for which each neuron is specialized. The orange columns are for neurons without active dendrites, while the blue columns are for neuron with active dendrites.

In summary, the experimental results in Table 4.2 demonstrate the effectiveness of the solution in mitigating the problem of catastrophic forgetting when the hyperparameters are properly chosen. Additionally, the neuron activity plot of Fig. 4.3 supports the hypothesis that active dendrites increase the sparsity of neural activity. Moreover, as shown in Fig. 4.4, the task-dependent gating mechanism ensures that each neuron is specialized to a subset of tasks, thereby reducing interference during training. This provides strong evidence that an SRM with a Rel-PSP kernel enhanced with active dendrites effectively mitigates the problem of catastrophic forgetting, offering a promising solution for continual learning in a TTFS-encoded network.

4.3. Hardware results

This section presents the results of the hardware implementation of the digital architecture presented in Section 3.2. Specifically, the hardware implementation realizes the neural network used for the software experiments of Section 4.2. The final objective of this section is to demonstrate the feasibility of a TTFS-encoded network with a hidden layer enhanced with active dendrites. To achieve this objective, this section is divided into subsections as follows. The first subsection explains the implementation and validation methodology. The second subsections provide the simulation waveforms of the implemented neural network, demonstrating a close match between software and hardware simulations. Finally, the last subsection provides a characterization of the hardware implementation in terms of power consumption, resource utilization, and latency.

4.3.1. Methodology

The methodology employed to implement the proposed architecture follows a bottom-up approach. Specifically, each component of the architecture illustrated in Fig. 3.5 has been individually implemented and validated with a testbench using behavioral simulations. Once these components were successfully validated, they were interconnected to construct a complete layer. Henceforth, following a behavioral evaluation

of the layer module, two layers were interconnected to implement the neural network architecture of Section 4.2.

Subsequently, the neural network architecture is synthesized and implemented to verify correct operations from a timing perspective. Following the implementation process, a timing analysis was performed to verify timing closure and explore the possibility of increasing the operating frequency.

Furthermore, to enable fast reconfigurability of the hardware, we employed a parameterized approach for each module. The top module, containing two interconnected layers, provides the flexibility to specify various parameters, including the pre-synaptic and post-synaptic neurons in each layer, the number of dendrites for each neuron, the threshold voltage, the precision of the synaptic and dendritic delays and the depth of the FIFOs. By following this approach, the dimensions of the registers and wires are automatically generated for different networks sizes.

To load synaptic weight parameters and dendritic delays in the Zynq-7020 BRAM, we used the Xilinx block memory generator IP. This block enables the initialization of the memory from *.coe* formatted files. To generate these files, we quantized the weight parameters and synaptic delays to a fixed-point precision during the training process. Subsequently, these parameters were converted into binary strings and written into a *.coe* file. This approach not only facilitates memory initialization during the design elaboration phase, providing initial values for simulation, but also enables the generation of a bitstream file to initialize the physical BRAMs of the FPGA.

Finally, to validate our hardware implementation with the software simulation, we first quantize the best-performing model of Table 4.1. Specifically, we quantized each synaptic weight to a 4-bit fixed-point precision, while each dendritic delay was quantized to an 8-bit fixed-point precision to ensure accuracy in the delay introduced by the active dendrites. Subsequently, we loaded the quantized parameters into the respective BRAMs and conducted behavioral, post-synthesis and post-implementation simulations using one sample images from the N-MNIST dataset to verify matching between the different phases.

It is important to note that while post-implementation simulations constitute a necessary step towards a proper evaluation of the proposed solution, a stronger validation would be offered by a full FPGA mapping where, image samples are stored in an SD card and loaded from the FPGA core to the programmable logic where the instance of the network resides. This further validation is left for future work.

4.3.2. Hardware neural network

The implemented neural network consists of two hardware layers:

- The first layer contains 1156 pre-synaptic neurons and 256 postsynaptic neurons, resulting in 256 NPU instances inside the neural cluster. Each NPU contains a down counter to load the dendritic delay.
- The second layer contains 256 pre-synaptic neurons and 5 post-synaptic neurons, resulting in 5 NPU instances inside the neural cluster.

This network was elaborated, synthesized, and implemented. An example image of the implemented design is provided in Fig. 4.5. To evaluate the matching between

hardware and software simulations, we conducted a behavioural simulation where three randomly selected images from the dataset are used as input to the network. In this context, each image represents a different task. Therefore, at the beginning of each simulation, the corresponding dendritic delay for the task was loaded into all 256 NPUs in the first layer. An example of the waveforms generated from the output neurons in response to these images is shown in Fig. 4.6.

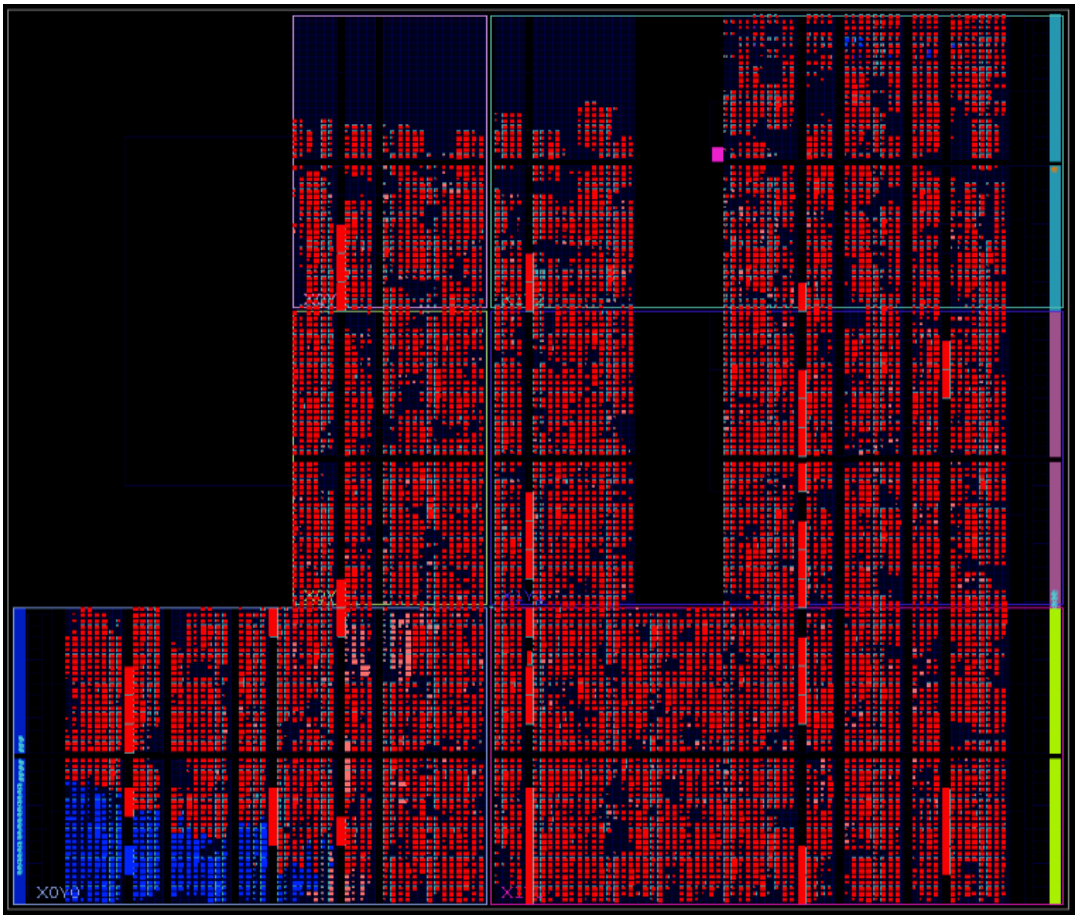


Figure 4.5: FPGA implementation view of the neural network. Red elements belong to the first layer, and blue elements belong to the second layer.

As explained in previous sections, in a TTFS-encoded network, the information is encoded in the spike time. Consequently, the classification output of a specific image is determined by the neuron that spikes first. For instance, if we attempt to classify the zero digit, then the spike time of the first neuron in the output layer should be the first to emit a spike. As can be seen from Fig. 4.6, there is a match between the input digit and the classified digit. For example, when classifying the zero digit, neuron with address zero spikes first, whereas when classifying the digit two, the neuron with address two spikes first. To further verify the match between hardware and software, Table 4.2 presents the time step of the spike events of each neuron of hardware and software simulations.

As can be seen in the table, the hardware provides a correct classification result. Most spike times match exactly or within 1-2 simulation steps. a small mismatch that likely results from the fact that in software simulations only the synaptic parameters

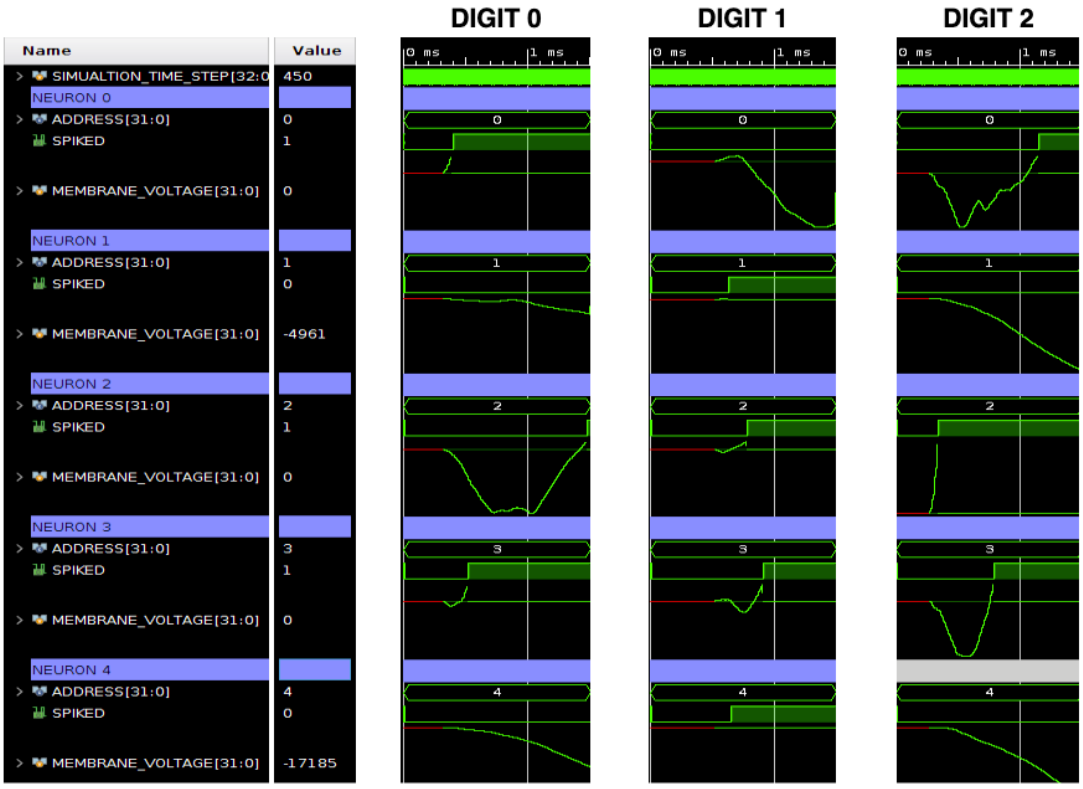


Figure 4.6: Behavioural simulation waveforms. Classification waveforms of the output neurons in response to the three randomly sample images. For each neuron the address, the spiked output and the membrane voltage are plotted.

Neuron	Digit 0		Digit 1		Digit 2	
0	117	117	X	X	341	402
1	X	X	187	187	X	X
2	442	374	233	235	101	100
3	154	152	271	270	232	232
4	X	X	195	195	X	X

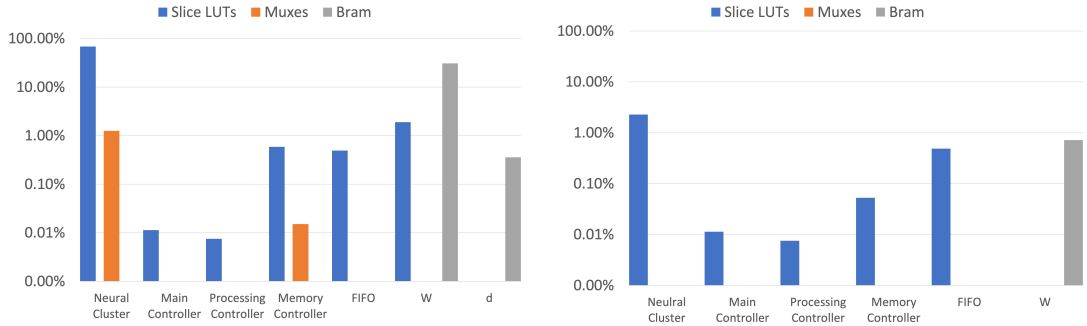
Table 4.2: Hardware-software correspondence in spike times for the neurons in the output layer. Grey entries represents the spike time of the hardware simulation. The X denotes a non-spiking (i.e. dead) neuron.

and dendritic delays are quantized, while the membrane potential is maintained with floating point-precision. Conversely, in the hardware simulations, the membrane potential uses fixed-point notation, thereby resulting in a difference in membrane potential integration between software and hardware. This also likely explains why the biggest discrepancies (see neuron 2 for digit 0 and neuron 0 for digit 2 in Table 4.2) manifest only for the late spike times, as such discrepancies accumulate until the change the network dynamics. Further investigation is left for future work.

4.3.3. Characterization

To obtain an accurate estimate of the power consumption of the proposed architecture under a typical operating condition. A post-synthesis simulation of a randomly-selected digit was used to create a *.saif* annotation file. This file provides the switching activity of each net in the design, allowing for an accurate estimate of the power consumption, assuming that the activity of this digit is representative of the full dataset. Specifically this simulation was conducted using the synthesized design at the post-implementation timing closure clock period $T_{clk} = 8\ ns$. From this analysis our networks is estimated to consume $122\ mW$ of dynamic power and $109\ mW$ of static power, for a total power consumption of $231\ mW$.

We evaluated the FPGA resources required to implement the suggested architecture by analyzing the Vivado utilization report. The proposed architecture utilizes a total of 39232 (74%) LUTs, 30169 (28%) FF, and 44.5 (32%) BRAMs. A detailed breakdown of the resource utilization for each component of a layer is provided in Fig. 4.7. Specifically, Fig. 4.7a illustrates the breakdown of resource utilization of the first layer, while Fig. 4.7b illustrates the utilization of the second layer.



(a) L1 utilization

(b) L2 utilization

Figure 4.7: Resource utilization by layer. Percentage of used resources by different modules within each layer of the network.

The estimate the average time required to classify an image in hardware without a deployed model we define the average classification time as

$$t_{avg} = T_{avg,out} \times t_{avg,step}, \tag{4.1}$$

where $T_{avg,out}$ is the average number of simulation steps required for the neurons in the output layer to emit the first spike, and $t_{avg,step}$ is the average time duration of a simulation step in hardware. The first can be estimated by running a software simulation across all images and measuring the average simulation step at which the neurons in the output layer fire their first spike, thereby determining the classification output. The second is largely determined by the number of pre-synaptic spikes processed by the hidden and output layers. This parameters is estimated as

$$t_{avg,step} = t_{min} \times (N_{hidden} + N_{output}), \tag{4.2}$$

where t_{min} is the time required to process one pre-synaptic spike in hardware, N_{hidden} is the average number of pre-synaptic spikes per time step in the hidden layer and N_{output} is the average number of pre-synaptic spikes per time in the output layer.

The last two parameters can be calculated from software simulations. On the other hand, the first parameter can be measured from post-implementation simulations by feeding a layer with a single pre-synaptic spike and measuring the time it takes for the time step to end. Consequently, by plugging Eq. (4.2) into Eq. (4.1) the average time required for the hardware to classify an image is defined as

$$t_{avg} = T_{out,avg} \times t_{min} \times (N_{hidden} + N_{output}). \tag{4.3}$$

These parameters were measured at the timing closure frequency $f_{clk} = 125\ MHz$, and are summarized in Table 4.3. The expected average time to classify an image of the dataset is $117.29\ \mu s$.

Parameter	**Value**
$T_{out,avg}$	284 steps
t_{min}	100 ns
N_{hidden}	3.94 spikes
N_{output}	0.19 spikes

Table 4.3: Measurements of parameters required to find t_{avg}.

5

Conclusion and Future Works

This thesis aims at evaluating the feasibility of a learning algorithm capable of mitigating the problem of catastrophic forgetting in TTFS-encoded spiking neural networks and designing its corresponding hardware accelerator following a neuromorphic approach. This work presents a novel TTFS-encoded model based on an SRM with a Rel-PSP kernel, enhanced with active dendrites. The key claims of this thesis are as follows:

- The biologically inspired concept of active dendrites provides a viable solution to mitigate the problem of catastrophic forgetting in TTFS-encoded networks.
- The inherent problem of dead neurons in TTFS encoded networks, when coupled with active dendrites, can be leveraged to create a gating mechanism that increases sparsity and gives rise to different subnetwork for different tasks.

The proposed model can successfully mitigate the problem of catastrophic forgetting when training the N-MNIST dataset sequentially, i.e. one digit class at a time. The final model consists of 1156 input neurons, 256 hidden active-dendrite neurons, and 5 output neurons, one for each digit. This model can be trained to classify the first five digits of the N-MNIST dataset sequentially, achieving an end-of-training test accuracy of 100% on all tasks. On the contrary, the same model without active dendrites achieves an end-of-training test accuracy of 23%, which is close to random-guessing performance. These results underscore the ability of active dendrites to effectively mitigate the problem of catastrophic forgetting using a TTFS-encoded neural network.

In addition to the theoretical novelty of the neuron model, we evaluated the feasibility of the proposed model on a digital hardware accelerator. To achieve this goal, we designed a custom hardware neural processing unit that can effectively leverage dendritic delays from active dendrites to delay the spike time in a task-dependent manner. This neural model was incorporated into an event-based layered architecture to form a TTFS encoded neural network. The proposed hardware implementation utilizes 74% of the LUTs, 28% of the FFs, and 32% of the BRAM available on a small-scale Xilinx Zynq-7020 SoC FPGA. The hardware network is expected to have an average classification time of $117\ \mu s$ and consumes $232\ mW$ at a clock frequency of $125\ MHz$.

However, this work also has several limitations. For instance, the use case employed to validate our proposed model leads to a trivial classification accuracy on

the test set. Additionally, our selected use case corresponds to a task-incremental continual-learning scenario, where the switch between tasks is known, a condition that is usually not met in the real word. From the hardware perspective, as the proposed implementation stopped with post-implementation simulations, it would benefit from further validation with a deployment in the programmable logic fabric of the FPGA.

To expand the breadth and depth of these claims, our work uncovers several venues for potential improvements and future research, as follows:

- **Improvements**:
  - **Task difficulty:** Increase the complexity of the continual learning scenario by testing the proposed solution on the split N-MNIST and permuted N-MNIST datasets.
  - **Model capacity:** Increase the number of tasks by an order of magnitude to verify the capacity of the model to retain information of a larger number of tasks. This improvement could also be tested with the sequential N-MNIST dataset by generating a random permutation of images for each new task.
  - **Task switching:** Explore a prototype method capable of dynamically inferring task switches from the data, similar to the approach used by Iyer et al. in [13].
  - **Quantization-aware training:** Create a quantization aware training setup, where parameters, activation functions and gradients are quantized.
  - **FPGA deployment:** Finalize the validation of the hardware design by testing the inference capabilities directly inside the FPGA board. To achieve this goal, one could load test samples of the dataset inside the SD card of the FPGA board. The SoC core should then load these images to the programmable logic of the FPGA where the model instance is deployed.
- **Future Work**:
  - **Lottery ticket hypothesis**: Although the task of classifying the N-MNIST digits sequentially only provides a simple proof of concept, it is worth highlighting that the proposed network can also be trained to 100% accuracy when the synaptic weight learning rate is zero. This surprising finding suggests that the network can learn the different tasks only by training the parameters of the active dendrites. The *lottery ticket hypothesis* suggests that, within a dense initialization weight matrix, there is a subset of sparse weights, which, if trained from scratch, can achieve the same accuracy performance as the dense network [67]. This hypothesis is further extended by Ramanujan et al. in [68], which suggests that the hidden weight matrix, if properly initialized, contains a set of sparse weights that can achieve impressive accuracy results without ever training the weights. Consequently, we hypothesize that active dendrites are capable of enhancing the activity of neurons attached to these sparse weights while reducing the activity of neurons that do not belong to the set of sparse winning weights. Thus, a promising avenue for future work is to investigate whether there is formal mathematical equivalence between the two mechanisms.

- **Online continual learning:** Ideally, the goal of neuromorphic computing is to design general-purpose machines that can learn online and process temporal data with short-to-long-term dependencies, while adapting to a dynamical environment where tasks are different and can change without prior notice. Recurrent spiking neural networks offer a viable solution to the problem of long-term dependencies [69]. Recently, different algorithms have been proposed to train these networks in a forward only fashion, i.e. without requiring a backward pass [70, 71, 72, 73, 74], thus allowing online network training. A potential avenue is to enhance these algorithms with the concept of active dendrites to enable a dynamical adaptation to different tasks.

In conclusion, this work has revealed a promising path for neuromorphic computing, where custom hardware based on TTFS-encoded neuron models enhanced with active dendrites can mitigate the problem of catastrophic forgetting. While our research journey provided a first compelling proof of concept, it also revealed promising avenues for future research.

Bibliography

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, A. M. Umar, O. U. Linus, H. Arshad, A. A. Kazaure, U. Gana, and M. U. Kiru, "Comprehensive review of artificial neural network applications to pattern recognition," *IEEE access*, vol. 7, pp. 158820–158846, 2019.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[3] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[5] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model.," in *Interspeech*, vol. 2, pp. 1045–1048, Makuhari, 2010.

[6] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *arXiv preprint arXiv:1904.05862*, 2019.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[8] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[9] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, pp. 109–165, Elsevier, 1989.

[10] C. Frenkel, D. Bol, and G. Indiveri, "Bottom-up and top-down neural processing systems design: Neuromorphic intelligence as the convergence of natural and artificial intelligence," *arXiv preprint arXiv:2106.01288*, 2021.

[11] 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, p. 638474, 2021.

[12] F. Ponulak and A. Kasinski, "Introduction to spiking neural networks: Information processing, learning and applications.," *Acta neurobiologiae experimentalis*, vol. 71, no. 4, pp. 409–433, 2011.

[13] A. Iyer, K. Grewal, A. Velu, L. O. Souza, J. Forest, and S. Ahmad, "Avoiding catastrophe: Active dendrites enable multi-task learning in dynamic environments," *Frontiers in neurorobotics*, vol. 16, p. 846219, 2022.

[14] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.

[15] N. Spruston, "Pyramidal neurons: dendritic structure and synaptic integration," *Nature Reviews Neuroscience*, vol. 9, no. 3, pp. 206–221, 2008.

[16] N. Takahashi, C. Ebner, J. Sigl-Glöckner, S. Moberg, S. Nierwetberg, and M. E. Larkum, "Active dendritic currents gate descending cortical outputs in perception," *Nature Neuroscience*, vol. 23, no. 10, pp. 1277–1285, 2020.

[17] T. Takeuchi, A. J. Duszkiewicz, and R. G. Morris, "The synaptic plasticity and memory hypothesis: encoding, storage and persistence," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 369, no. 1633, p. 20130288, 2014.

[18] S. R. y Cajal, "Histologie du systeme nerveux de l'homme et des vertebre´ s ii," *(No Title)*, 1909.

[19] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.

[20] W. Gerstner, A. K. Kreiter, H. Markram, and A. V. Herz, "Neural codes: firing rates and beyond," *Proceedings of the National Academy of Sciences*, vol. 94, no. 24, pp. 12740–12741, 1997.

[21] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[22] R. S. Johansson and I. Birznieks, "First spikes in ensembles of human tactile afferents code complex spatial fingertip events," *Nature neuroscience*, vol. 7, no. 2, pp. 170–177, 2004.

[23] E. M. Izhikevich, N. S. Desai, E. C. Walcott, and F. C. Hoppensteadt, "Bursts as a unit of neural information: selective communication via resonance," *Trends in neurosciences*, vol. 26, no. 3, pp. 161–167, 2003.

[24] H. G. Eyherabide, A. Rokem, A. V. Herz, and I. Samengo, "Bursts generate a non-reducible spike-pattern code," *Frontiers in Neuroscience*, vol. 3, p. 490, 2009.

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

[26] J. O'Keefe, "Hippocampus, theta, and spatial memory," *Current opinion in neurobiology*, vol. 3, no. 6, pp. 917–924, 1993.

[27] W. Gerstner, “Time structure of the activity in neural network models,” *Physical review E*, vol. 51, no. 1, p. 738, 1995.

[28] R. E. Burke, “Composite nature of the monosynaptic excitatory postsynaptic potential.,” *Journal of Neurophysiology*, vol. 30, no. 5, pp. 1114–1137, 1967.

[29] K. Frank, “Basic mechanisms of synaptic transmission in the central nervous system,” *IRE Transactions on Medical Electronics*, no. 2, pp. 85–88, 1959.

[30] P. Malík, “High throughput floating point exponential function implemented in fpga,” in *2015 IEEE Computer Society Annual Symposium on VLSI*, pp. 97–100, IEEE, 2015.

[31] M. Zhang, J. Wang, J. Wu, A. Belatreche, B. Amornpaisannon, Z. Zhang, V. P. K. Miriyala, H. Qu, Y. Chua, T. E. Carlson, *et al.*, “Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 33, no. 5, pp. 1947–1958, 2021.

[32] E. O. Neftci, H. Mostafa, and F. Zenke, “Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks,” *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[33] S. M. Bohte, J. N. Kok, and H. La Poutre, “Error-backpropagation in temporally encoded networks of spiking neurons,” *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[34] Y.-C. Hsu, Y.-C. Liu, A. Ramasamy, and Z. Kira, “Re-evaluating continual learning scenarios: A categorization and case for strong baselines,” *arXiv preprint arXiv:1810.12488*, 2018.

[35] F. Zenke, B. Poole, and S. Ganguli, “Continual learning through synaptic intelligence,” in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 3987–3995, PMLR, 06–11 Aug 2017.

[36] I. J. Goodfellow, M. Mirza, X. Da, A. C. Courville, and Y. Bengio, “An empirical investigation of catastrophic forgeting in gradient-based neural networks,” in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.

[37] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[38] R. V. W. Putra and M. Shafique, “Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1057–1062, IEEE, 2021.

[39] P. Panda, J. M. Allred, S. Ramanathan, and K. Roy, “Asp: Learning to forget with adaptive synaptic plasticity in spiking neural networks,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 51–64, 2017.

[40] J. M. Allred and K. Roy, “Controlled forgetting: Targeted stimulation and dopaminergic plasticity modulation for unsupervised lifelong learning in spiking neural networks,” *Frontiers in neuroscience*, vol. 14, p. 7, 2020.

[41] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[42] N. Y. Masse, G. D. Grant, and D. J. Freedman, “Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization,” *Proceedings of the National Academy of Sciences*, vol. 115, no. 44, pp. E10467–E10475, 2018.

[43] I. Hammouamri, T. Masquelier, and D. Wilson, “Mitigating catastrophic forgetting in spiking neural networks through threshold modulation,” *Transactions on Machine Learning Research*, 2022.

[44] P. P. Gao, J. W. Graham, W.-L. Zhou, J. Jang, S. Angulo, S. Dura-Bernal, M. Hines, W. W. Lytton, and S. D. Antic, “Local glutamate-mediated dendritic plateau potentials change the state of the cortical pyramidal neuron,” *Journal of neurophysiology*, vol. 125, no. 12, pp. 23–42, 2021.

[45] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[46] L. Lapicque, “Recherches quantitatives sur l’excitation electrique des nerfs,” *J Physiol Paris*, vol. 9, pp. 620–635, 1907.

[47] R. Golden, J. E. Delanois, P. Sanda, and M. Bazhenov, “Sleep prevents catastrophic forgetting in spiking neural networks by forming a joint synaptic weight representation,” *PLOS Computational Biology*, vol. 18, no. 11, p. e1010628, 2022.

[48] G.-q. Bi and M.-m. Poo, “Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type,” *Journal of neuroscience*, vol. 18, no. 24, pp. 10464–10472, 1998.

[49] C. Frenkel, J.-D. Legat, and D. Bol, “Morphic: A 65-nm 738k-synapse/mm ^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning,” *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 5, pp. 999–1010, 2019.

[50] K. Boahen, “Point-to-point connectivity between neuromorphic chips using address events,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000.

[51] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[52] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, "The brainscales-2 accelerated neuromorphic system with hybrid plasticity," *Frontiers in Neuroscience*, vol. 16, p. 795876, 2022.

[53] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm ^212.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 1, pp. 145–158, 2018.

[54] S. Yin, S. K. Venkataramanaiah, G. K. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, and J. sun Seo, "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations," 2017.

[55] K. T. N. Chu, B. Amornpaisannon, Y. Tavva, V. P. K. Miriyala, J. Wu, M. Zhang, H. Li, T. E. Carlson, *et al.*, "You only spike once: Improving energy-efficient neuromorphic inference to ann-level accuracy," *arXiv preprint arXiv:2006.09982*, 2020.

[56] F. Corradi, G. Adriaans, and S. Stuijk, "Gyro: A digital spiking neural network architecture for multi-sensory data analytics," in *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*, pp. 9–15, 2021.

[57] A. Sankaran, P. Detterer, K. Kannan, N. Alachiotis, and F. Corradi, "An event-driven recurrent spiking neural network architecture for efficient inference on fpga," in *Proceedings of the International Conference on Neuromorphic Systems 2022*, ICONS '22, (New York, NY, USA), Association for Computing Machinery, 2022.

[58] C. Frenkel, J.-D. Legat, and D. Bol, "A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2020.

[59] S. Widmer, M. Kossel, G. Cherubini, S. Woźniak, P. A. Francese, A. Stanojevic, M. Brändli, K. Frick, and A. Pantazi, "Design of time-encoded spiking neural networks in 7nm cmos technology," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[60] H. Li, X. Ye, A. Imakura, and T. Sakurai, "Ensemble learning for spectral clustering," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 1094–1099, 2020.

[61] F. T. Zohora, V. Karia, A. R. Daram, A. M. Zyarah, and D. Kudithipudi, “Metaplasticnet: Architecture with probabilistic metaplastic synapses for continual learning,” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2021.

[62] V. Karia, F. T. Zohora, N. Soures, and D. Kudithipudi, “Scolar: A spiking digital accelerator with dual fixed point for continual learning,” in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1372–1376, IEEE, 2022.

[63] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” 2019.

[64] C. Posch, D. Matolin, and R. Wohlgenannt, “A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds,” *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2010.

[65] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, “Training spiking neural networks using lessons from deep learning,” *arXiv preprint arXiv:2109.12894*, 2021.

[66] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” 2015.

[67] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” 2019.

[68] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, “What’s hidden in a randomly weighted neural network?,” 2020.

[69] C. Frenkel and G. Indiveri, “ReckOn: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales,” in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, IEEE, feb 2022.

[70] 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, no. 1, p. 3625, 2020.

[71] T. Bohnstingl, S. Woźniak, A. Pantazi, and E. Eleftheriou, “Online spatio-temporal learning in deep neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[72] B. Yin, F. Corradi, and S. M. Bohte, “Accurate online training of dynamical spiking neural networks through forward propagation through time,” 2022.

[73] T. Ortner, L. Pes, J. Gentinetta, C. Frenkel, and A. Pantazi, “Online spatio-temporal learning with target projection,” in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–5, 2023.

[74] F. M. Quintana, F. Perez-Peña, P. L. Galindo, E. O. Neftci, E. Chicca, and L. Khacef, “Etlp: Event-based three-factor local plasticity for online learning with neuromorphic hardware,” 2023.