Once Upon a Spike Time: Exploring Spiking Neurons and Perspective Applications of Neuromorphic Solutions

PhD Thesis

Davide Liberato Manna

A thesis submitted for the degree of

Doctor of Philosophy

Neuromorphic Sensor Signal Processing Lab

Centre for Signal and Image Processing

Department of Electronic and Electrical Engineering

University of Strathclyde

Glasgow

2025

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Signed: Davide Liberato Manna
Date: February 23, 2025

Acknowledgement

I would like to express my sincerest appreciation to my supervisor, Dr Gaetano Di Caterina, for believing in me from the first moment, for his constant presence, and for his support. His guidance throughout my studies has been invaluable as well as his understanding in times of need.

I would also like to thank my friend and mentor Dr Paul Kirkland for the endless hours spent chatting and hearing me ranting, and for making me feel at home. Half of the things in this thesis wouldn't have been achieved without his presence. A big thanks goes to Alex Vicente Sola and David Vint, my PhD colleagues. Together with Paul, they made my day-to-day great and constituted my lab family from day 0.

Thanks to Dr Trevor J. Bihl and the Air Force Research Laboratory for sponsoring my PhD (Grant Number FA8655-20-1-7037), which would not have been possible otherwise. Dr Bihl has been a constant presence throughout my PhD, providing valuable feedback, and good laughs too.

The biggest thanks goes to my wife, Serena, for her love, her care, and her patience in the countless times she had to endure my complaints and desperation. All of this, whilst doing a PhD herself and whilst gifting me with our daughter Maze Eva, who has been the true push for me to write up this thesis.

Finally, thanks to my friends (old and new) in Glasgow, Thomas, Emily, Jaqueline, Tiziano, Alessia, Damaris and Michele. Thanks for the support, for not asking too many questions about my PhD and for celebrating with me.

Abstract

Despite their outstanding achievements in a number of different fields, conventional neural network-based solutions often fall short in terms of concrete applicability to edge devices and systems with limited autonomy or computational resources. This is due to the intensive computations that are required to process input data and the size of modern neural networks, which also test the memory requirement of a given setup. Neuromorphic (NM) computing offers a shift in this paradigm by enabling low-power Artificial Intelligence (AI) through sparse and localized computations, and low latency thanks to its asynchronicity. Spiking Neural Networks (SNN), with spiking neurons at their core, represent the algorithmic foundation of Neuromorphic AI. In recent years, the research on SNNs has been highly proactive with several advancements being proposed in various contexts, in an effort to try and bridge the performance gap with conventional approaches that has traditionally characterized the field of NM computing. Whilst research efforts are showing promising results, there is still a lack of shared foundational knowledge and understanding of the interplay between different components, particularly the spiking neuron models. Furthermore, a reason for debate is given by what represents a good task for the SNNs to be evaluated given that more traditional ones might not fully highlight the representational abilities of SNNs. As such, new directions of research are always sought, where new tasks that could benefit from the use of SNNs' feature extraction abilities are explored. This thesis aims to address the above points as a means to help advance the overall knowledge in the field of Neuromorphic computing with SNNs. The starting point of the investigations resides in the understanding of the importance of spiking neurons in an NM machine learning (ML) pipeline. In the neuroscience literature, spiking neuron models are extensively discussed as they are the means by which scientists model the human brain. Such mathematical models determine how incoming information affects the internal state of a neuron in relation to one or more variables, and regulate the emission of spikes, the signals that neurons use to communicate information. The same level of attention is not nor-

mally paid when employing neuron models in engineering pipelines and, often, the simplest model, the Leaky Integrate-and-Fire (LIF), is used for its simplicity and efficiency. Starting from a selection of three relatively simple models, this thesis presents a study aimed at highlighting whether the choice of any neuron model is more appropriate with respect to another one. This is tested within a simple framework where an SNN is trained with Spike Time-Dependent Plasticity (STDP), a biologically inspired unsupervised learning rule. The tasks presented to the SNN are image classification tasks based on NM datasets of increasing difficulty. The study reveals that higher levels of complexity in the neuronal dynamics can in fact prove beneficial where the complexity of the temporal features in the data is also higher. The thesis continues with an exploration of a possible approach to a field that is not often considered for NM applications: that of time series forecasting. The proposed approach encompasses two main aspects: a novel neuron population encoding system, and two novel bespoke loss functions. The encoding system leverages concepts from the differencing transform used in time series analysis and gets inspiration from neuromorphic vision sensors. In this way, the encoded signal is not only rendered more stationary and amenable to processing but it is also shown to approximate the derivative of the signal itself. The proposed loss functions build upon biological concepts and from the knowledge about the encoding step. The overall solution comprising the encoding system and one of the proposed loss functions is shown to outperform the reference system, thereby demonstrating the potential of SNNs to be applied in this domain. Finally, a novel solution to surface Electromyography (sEMG) gesture classification is presented. sEMG is a crucial technology frequently utilized in health-related applications, including prosthetics, where the advantages brought by NM engineering could be highly beneficial. The solution comprises the use of Resonate-and-Fire (RF) neurons, a type of neuron that has been shown to approximate a Short Time Fourier Transform (STFT), to encode EMG signals into a spike-frequency domain whilst performing filtering on the unwanted frequencies. This is paired with a novel decoding approach that leverages a convolutional layer to transform the signal back into the temporal domain, thereby enabling hyperparameter optimization on the encoding neurons. The overall solution is tested on a challenging dataset and is shown to outperform reference works on the same task. This final piece of research not only underscores the importance of the selection of appropriate spiking neurons for a given task but also how this can enable closing the performance gap between NM computing and conventional methods whilst maintaining the advantages of neuromorphic technologies.

Contents

Acronyms

AdEx	Adaptive Exponential Integrate-and-Fire.
ANN	Artificial Neural Network.
CNN	Convolutional Neural Network.
DL	Deep Learning.
EIF	Exponential Integrate-and-Fire.
EMG	Electromyography.
HH	Hodgkin-Huxley's.
IoT	Internet of Things.
IZ	Izhikevich's.
LIF	Leaky Integrate-and-Fire.
ML	Machine Learning.
NM	Neuromorphic.
NN	Neural Network.
QIF	Quadratic Integrate-and-Fire.
ReLU	Rectified Linear Unit.
RF	Resonate-and-Fire.
SNM	Spiking Neuron Model.
SNN	Spiking Neural Network.

STDP	Spike-Timing-Dependent Plasticity.
STFT	Short-Time Fourier Transform.

Chapter 1

Introduction

Artificial Intelligence (AI) and Machine Learning (ML) have seen rapid advancements, profoundly impacting numerous domains by offering intelligent solutions to complex problems [1]. This renaissance in AI research is largely credited to significant breakthroughs in Deep Learning (DL), particularly highlighted by the success of Convolutional Neural Networks (CNNs) in the 2012 ImageNet Challenge [2, 3]. This event, which involved the classification of images into 1000 distinct categories, showcased CNNs' ability to achieve and even surpass human-level performance in image recognition tasks.

The influence of AI and ML extends across various fields [1], demonstrating the versatility and potential of these technologies. In the realm of computer vision, deep learning techniques have revolutionized image and video analysis, enabling advancements in facial recognition, autonomous driving, and medical imaging [4]. Natural language processing (NLP) has similarly benefited, with AI models now excelling in tasks such as language translation, sentiment analysis, and conversational agents [5]. Moreover, AI applications in robotics have enhanced capabilities in navigation, manipulation, and human-robot interaction, further illustrating the transformative impact of these technologies [6].

As AI technologies mature, there is an increasing drive for their integration into everyday devices, leading to the proliferation of the Internet of Things (IoT) and autonomous devices [7]. The vision of ubiquitous AI involves embedding intelligence into a myriad of interconnected devices, from household appliances to industrial machinery, enabling them to communicate, make decisions, and optimize their operations autonomously. This integration promises to enhance efficiency, reduce costs, and improve the quality of life across various sectors, including healthcare, transportation, and smart cities [7].

However, the widespread deployment of AI and IoT presents several chal-

lenges. One of the primary concerns is the need for energy-efficient computing solutions. Traditional AI models, particularly deep learning networks, require substantial computational power and energy [8], which is not sustainable for IoT devices that often operate under strict power constraints. Additionally, the vast amount of data generated by IoT devices necessitates real-time processing and decision-making [9], which further exacerbates the demand for efficient AI solutions. In light of these challenges, Neuromorphic (NM) computing paradigms offer a transformative shift in the development and deployment of AI applications [10]. Thanks to the underlying low-power and low-latency advantages that it offers, NM computing-based solutions hold the potential to enable AI-backed IoT devices capable of performing on-the-fly operations whilst maintaining power consumption levels that permit long-term autonomy.

1.1 Research Motivations

The motivation behind the research presented in this thesis is rooted in the desire to explore and expand the boundaries of Neuromorphic computing. Traditional deep learning approaches, while powerful, often require substantial computational resources and energy consumption [10, 11]. Neuromorphic computing, inspired by the brain's architecture and functioning, offers a promising alternative by emulating neural processes through spiking neurons and specialized hardware [10]. Spiking neurons represent the core of a Spiking Neural Network (SNN), which in turn is the key component of Neuromorphic AI systems. Spiking neurons are modelled after their biological counterparts in the human brain and have the peculiar capability to process information in the time dimension. This is due to the fact that, as per their mathematical model, they hold an internal state that varies through time, and every input that is received perturbs this very state in a way that depends on the timing of the input itself, i.e. when it was received [12]. As a result of the evolution of their internal state, spiking neurons emit so-called spikes whenever a set threshold is reached and communicate, in this way, information with the rest of the neural network [13, 14]. Because spikes are not emitted at every point in time, but only once the threshold is reached, on top of enabling time-based computations, spiking neurons actuate a sparse communication system, thereby reducing the power requirements of the system, which in more conventional cases would suffer from having to constantly propagate information through the network, even when information is potentially zero [10, 15]. The above is enabled by brain-inspired NM hardware, which differs from

the conventional von Neumann architectures and provides the physical substrate onto which SNNs can be implemented at their best. As a matter of fact NM chips enable low-power and asynchronous information processing which is typical of the human brain, thus providing the perfect pairing for the SNNs [15].

The exploration into SNNs and their applications aims to address several key points. Firstly, there is a growing need for energy-efficient computing solutions, particularly for embedded autonomous systems and IoT devices where power constraints are critical [7]. Secondly, the unique properties of the diverse spiking neuron models present in the literature [12, 16] highlight the variety of contexts in which either type of model can be beneficial to a given system. Each model has the potential to process and extract temporal information in a different way, thus making it more or less suitable in specific use cases [17]. Lastly, expanding the scope of the applications of NM computing solutions can lead to the design of more robust and adaptive AI systems for a number of different fields, including time series forecasting and prosthetics [10].

To address the above, this thesis delves into the potential of employing different types of spiking neurons, and into the exploration of avenues that live both within and beyond the conventional neuromorphic application domains. By investigating various spiking neuron models, encoding systems, and learning rules, the aim is to contribute to the growing body of knowledge in this emerging field and highlight the advantages and possibilities associated with neuromorphic computing. By exploring possible new avenues of applications, the goal is on one hand to broaden the spectrum of domains that could benefit from the use of NM-based systems, and on the other hand to find valid tasks that can be used to evaluate the performance of an SNN, which is still a very active field of research [18]. The above is achieved by means of a progressive approach that begins by looking at a foundational component of an SNN, the spiking neuron, to understand the capabilities of different models with varying complexity to be more or less apt at solving a task. This evolves into the use of a relatively simple form of a population of spiking neurons able to encode the information present in an input signal in a way that leverages concepts from the classical processing methods for time series, and concepts from the operational design of event-based vision sensors. These are used in combination with two novel loss functions to construct a system for the prediction of time series. Time series, albeit not normally considered a conventional NM task due to the data being collected using non-NM hardware, naturally embed the concept of time and can therefore benefit from the computational paradigms brought by NM technologies. This thesis focuses

on energy load forecasting using two different types of datasets, however, the developed algorithms are task-agnostic, and can in principle be applied to any type of time series forecasting problem. Finally, the employment of the Resonate-and-Fire (RF) spiking neurons [19] for the encoding of Electromyography (EMG) signal into a spike-frequency domain not only serves as a further demonstration of the importance of using the correct spiking neuron model for a given task, but also that bridging the gap between conventional ML and NM-based solution in terms of accuracy is an attainable feat. As a matter of fact, despite providing sensible improvements in terms of power efficiency, SNNs have often fallen short in accuracy when compared against more conventional systems [10]. However, as shown later on in this thesis, this gap can be closed by adopting a suitable combination of spiking neuron models such that their information encoding and feature extraction capability is optimized for the task at hand. The application domain for the above solution is limited to the EMG classification of gestures. Nevertheless, the algorithm is once again agnostic of the data, if not for engineered hyperparameters. The algorithm can in fact be in principle applied to any data source that features information in the frequency domain. One relatable example is provided by electroencephalogram (EEG) signals, which are collected similarly to EMG signals. Furthermore, thanks to the developed decoding scheme which is able to transcode spike-frequency encoded information into time-based discretized signals, it is possible to conceive other potential applications that go beyond the scope of classification models and delve into that of generative models by providing a way to generate new signals starting from a sequence of spikes.

1.2 Original Contributions

The studies carried out as part of this thesis and briefly anticipated in the previous section led to a number of novel contributions to the NM scientific community, that are here summarized.

- **Design of a testing framework for spiking neuron models, and analysis of the advantages and disadvantages of using more or less complex models.** The study involves the use of three carefully selected models within a relatively simple context. The developed SNN is trained entirely through the Spike-Timing-Dependent Plasticity (STDP) learning rule and an in-depth study is carried out to understand the interplay between the use of certain neuron models and STDP (Chapter 4). This has been made possible thanks to the development of an expansion to Spyke-

Torch [20], a framework to develop SNNs. This extension revolves around the implementation of a number of spiking neuron models and is reported in Appendix A for reference.

- **Development of a novel encoding scheme inspired by the concept of differencing for time series signals and by Neuromorphic vision sensors**. This differencing encoding is shown to approximate the derivative of a signal and is employed to approach the time series forecasting problem, which requires some sort of translation of the information into spikes for the SNN to be able to process it. The encoding system is shown to enable learning firstly by initial tests on dummy sinusoid signals, and then on data taken from electricity load forecasting datasets [21–23] (Chapter 5).

- **Definition of two novel loss functions for the SLAYER [24] learning rule**. These, in combination with the encoding presented at the previous point, are thoroughly tested and combined with the standard learning rule for SLAYER. The DecodingLoss is shown to consistently allow achieving higher levels of performance than the other options (Chapter 5).

- **Design of a novel EMG signal classification pipeline based on the use of Resonate-and-Fire neurons for the encoding of the information.** A bank of RF neurons senses an EMG signal and produces spikes whenever certain frequencies are found, thus creating a spike-frequency representation. Such a pipeline allows to achieve extremely high accuracy levels on a very challenging EMG gestures dataset [25], enough to performing better than other works from both the NM and conventional DL literature (Chapter 6).

- **Design of a novel decoding methodology for RF neuron encoding based on the use of convolutional layers to transform a signal in the spike-frequency domain back into the original domain.** Such a decoding system allows the encoded-decoded signal to be compared against its original version, thus not only allowing hyper-parameter optimization, but also enhancing interpretability (Chapter 6).

1.3 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 covers the relevant literature from the conventional Deep Learning field and the NM computing domain. From the DL point of view, this encompasses a brief history of Artificial Neural Network (ANN)s and Convolutional Neural Network (CNN), and highlights the importance of activation and loss functions. The backpropagation algorithm is also presented as being at the core of an NM learning system employed later. From the NM computing point of view, the chapter reports a number of different spiking neuron models that are relevant to this work but also touches on concepts like neural coding and unsupervised/supervised learning systems. The chapter concludes with a review of the most prominent software frameworks for the development of SNNs. Chapter 3 delves into two relevant applications, time series forecasting and EMG gesture classification, that have been a target of study in this thesis. The Chapter provides a brief overview of the classical methodologies to approach each application so as to provide a contextual background and finally reviews the literature that is more pertinent to the performed studies. Chapter 4 details a study of the performance differences of three single-variable spiking neurons. The spiking neurons are tested within relatively simple image classification scenarios and insights regarding their differences are discussed. Chapter 5 presents an approach to the time series forecasting problem. The approach leverages concepts from the differencing transform and event vision sensors to design an encoding system for time series data. The Chapter also introduces the two novel loss functions that are employed and discussed in the study. Chapter 6 describes a proposed solution to the EMG gesture classification problem which employs RF spiking neurons. The proposed pipeline leverages the encoding and the decoding of the information from the EMG signals to build a classification framework that is able to compete with and surpass the current state of the art. Finally, Chapter 7 summarises the most important findings from the studies carried out in this thesis and outlines interesting research directions to be explored in the future.

1.4 List of Publications

1. **D. L. Manna**, A. Vicente-Sola, P. Kirkland, T. J. Bihl, and G. Di Caterina, "Simple and complex spiking neurons: perspectives and analysis in a simple STDP scenario," Neuromorphic Computing and Engineering, vol. 2, no. 4, p. 044009, 2022.

2. A. Vicente-Sola, **D. L. Manna**, P. Kirkland, G. Di Caterina, and T. Bihl, "Keys to accurate feature extraction using residual spiking neural networks," Neuromorphic Computing and Engineering, vol. 2, no. 4, p. 044001, 2022.

3. P. Kirkland, **D. Manna**, A. Vicente, and G. Di Caterina, "Unsupervised spiking instance segmentation on event data using STDP features," IEEE Transactions on Computers, vol. 71, no. 11, pp. 2728–2739, 2022.

4. **D. L. Manna**, A. Vicente-Sola, P. Kirkland, T. J. Bihl, and G. Di Caterina, "Frameworks for snns: a review of data science-oriented software and an expansion of SpykeTorch," in International Conference on Engineering Applications of Neural Networks, pp. 227–238, Springer Nature Switzerland Cham, 2023.

5. P. Chaudhari, A. Vicente-Sola, A. Basu, **D. L. Manna**, P. Kirkland, and G. Di Caterina, "Sign language recognition using spiking neural networks," Procedia Computer Science, vol. 235, pp. 2674–2683, 2024.

6. T. J. Bihl, P. Farr, G. Di Caterina, P. Kirkland, A. Vicente Sola, **D. L. Manna**, J. Liu, and K. Combs, "Exploring spiking neural networks (snn) for low size, weight, and power (swap) benefits", 57th Annual Hawaii International Conference on System Sciences (HICSS24), 2024.

7. **D. L. Manna**, A. Vicente-Sola, P. Kirkland, T. J. Bihl, and G. Di Caterina, "Time Series Forecasting via Derivative Spike Encoding and Bespoke Loss Functions for Spiking Neural Networks," Computers, Vol 13, no. 8, p. 202, 2024.

Chapter 2

Neuromorphic Computing and Neural Networks

2.1 Conventional Neural Networks

Since the breakthrough of Convolutional Neural Networks in 2012 [3] within the ImageNet Challenge [2] — a prominent image classification competition featuring 1000 classes such as dog breeds, cars, and birds — Neural Networks (NNs) and CNNs, in particular, have soared in popularity, initiating a renewed wave in artificial intelligence known as Deep Learning. This surge has led to an exponential increase in publications annually and the inclusion of deep learning topics in most engineering conferences. However, the concept of Neural Networks is not new, tracing back to the work of McCulloch and Pitts in the 1940s [26]. Their research was inspired by the brain's architecture, employing a network of connected neurons to perform tasks like classification and recognition. The backpropagation learning mechanism, crucial to deep learning, was developed by several researchers in the 1960s [27] and later implemented on computers in the 1970s. It was first applied to neural networks in 1974 [28].

The rise of neural networks, especially CNNs, was significantly propelled by the advent of Graphics Processing Units (GPUs) in the late 2000s [29]. GPUs, with their massively parallel processing capabilities, can process data between 4 and 70 times faster than Central Processing Units (CPUs) [27, 30], drastically accelerating the learning process. This computational power paired with large datasets enabled neural networks to surpass human performance in the ImageNet challenge by 2015 [31]. Following this initial success in image classification, ANNs and, more in general, DL-based methods have been proven to be extremely well suited for a number of different applications. In this thesis, the

concept of ANN stands at the core of the performed studies, although featuring characteristics that differ from the more conventional ones found in the literature. Nevertheless, the basic mechanics of NNs interconnections, activation functions and backpropagation still apply in a certain way.

For this reason, this section briefly reviews neural networks and their main characteristics. Section 2.1.1 covers Artificial Neural Networks and Convolutional Neural Networks, focussing on the types of layers that differentiate them. Section 2.1.2 introduces activation functions and gives examples of possible conventional ones. Section 2.1.4 discusses the backpropagation algorithm. Section 2.1.3 presents the concept of cost functions and their importance within a DL pipeline.

2.1.1 ANNs and CNNs

As previously anticipated, the concept of neural networks dates back to the early 1940s with Warren McCulloch and Walter Pitts' pioneering work, which laid the groundwork for artificial neural networks [26]. They introduced a simplified model of a biological neuron, known as the McCulloch-Pitts neuron (Eq. (2.1)), which used a binary threshold logic to simulate the basic functioning of neural activity. This model set the stage for subsequent developments in the field of artificial intelligence and neural computation.

$$y = H\left(\sum_{i=1}^{n} x_i\right) \tag{2.1}$$

Following the McCulloch-Pitts model, Frank Rosenblatt introduced the Perceptron in 1958 [32], which became one of the first instances of an ANN capable of learning. The perception, defined by Eq. (2.2), demonstrated the potential of NNs in pattern recognition tasks but also highlighted limitations, such as its inability to solve non-linearly separable problems. The latter issue was addressed by the introduction of multi-layer perceptrons (MLPs), non-linear activation functions, and the backpropagation algorithm in the 1980s, pioneered by Rumelhart, Hinton, and Williams [33]. These advancements allowed for deeper networks and the learning of more complex patterns.

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b) \tag{2.2}$$

While traditional ANNs had made significant strides, they encountered limitations in processing structured grid-like data, such as images. Yann LeCun and

his colleagues addressed these challenges by developing the first CNNs in the late 1980s and early 1990s. LeCun's 1998 paper on LeNet-5 [34], a CNN architecture designed for handwritten digit recognition, demonstrated the power of CNNs in handling spatial hierarchies through local receptive fields, shared weights, and spatial subsampling (pooling).

The unique architecture of CNNs, with their convolutional layers and pooling layers, enabled the automatic extraction of spatial features from images, making them particularly effective for image-related tasks. The introduction of large-scale datasets such as ImageNet and advancements in GPU technology in the 2010s led to a renaissance in CNN research and applications [35]. Notable architectures like AlexNet (2012) [3], VGGNet (2014) [36], and ResNet (2015) [37] pushed the boundaries of image classification accuracy and deep learning capabilities.

In practical terms, what differentiates an ANN from a CNN is mainly the connectivity pattern found within the network. An ANN is normally considered to be composed of a series of so-called fully connected layers. Fully connected, or dense, layers do exactly what the name suggests: connect every neuron in one layer to every input, so that given an N-dimensional input and an M-dimensional layer (i.e. a layer with M neurons), the total number of connections is NxM. Conversely, a so-called Convolutional (Conv) layer is a connection scheme that performs a cross-correlation operation (differently from what the name suggests) [35]. Conv layers are characterized by a set of filters (kernels) and a stride. The kernels are cross-correlated with the input using the stride as the quantity of shift across the input at each computation. Theoretically, the kernels can be of any dimension, however, the most common cases see them used as 2D or 3D kernels. The result of utilizing a Conv layer is the production of a feature map given by the sum of the element-wise (Hadamard) product between the elements in the input and in the kernel. Fig. 2.1 provides a visual example of the operation performed as part of the convolutional layer, while the following is the formulation of the Conv layer operation for a 2-dimensional case:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} K_{m,n} \cdot I_{i-m,j-n} + b, \tag{2.3}$$

where I is the input, K is the kernel, i and j represent the stride indices, m and n represent the coordinates of each element within the kernel, and b is the bias.

The size of the feature map depends on the initial input size, the kernel size, the stride, and the presence of padding (placeholder values, normally 0, placed

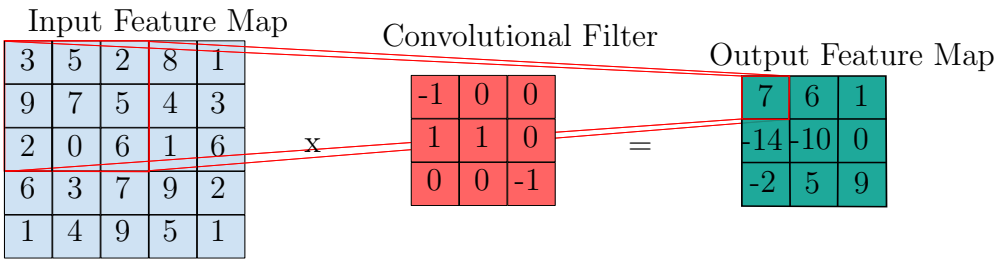


Fig. 2.1: Simple Visual Example of Convolutional Layer with kernel 3x3 and stride 1. The input (5x5 matrix) is parsed by the 3x3 kernel starting from the top left corner and moving right and downwards. The output feature map contains the sum of the element-wise product between the input in a given location and the kernel. Notably, the output feature map is reduced in size.

around the input), and can be calculated according to the following formula:

$$n_{out} = \frac{n_{in} + 2p - k}{s} + 1, \tag{2.4}$$

where n_{out} is the output size along one dimension, n_{in} is the size of the input along the same dimension, p is the amount of padding (calculated times 2 to account for padding at the beginning and at the end), and s is the stride. Therefore, where a fully connected layer with M neurons would produce an output that is of size M, a 2D Conv layer with M kernels would produce an output that is MxH_{out}xW_{out}. In the limit case of a Conv layer with kernel size equal to the input size, the output of the Conv layer and of the Fully connected layer would be the same in size.

2.1.2 Activation Functions and Dropout

As briefly discussed in the previous section, in the development of Artificial Neural Networks the use of non-linear activation functions was a critical point in allowing to solve non-linearly separable problems. Functions such as the sigmoid or the tanh were necessary to break the linearity of Perceptrons and to increase the depth of the networks [35, 38], thus rendering the solution of increasingly complex tasks attainable. A further step was possible with the introduction of the Rectified Linear Unit (ReLU), which made it possible to avoid issues such as slow convergence and vanishing gradient, problems in which the previously mentioned functions incurred. The ReLU is defined in the following way:

$$\sigma(x) = max(0, x) = \begin{cases} x & \text{if} \quad x \geq 0, \\ 0 & \text{if} \quad x < 0. \end{cases} \tag{2.5}$$

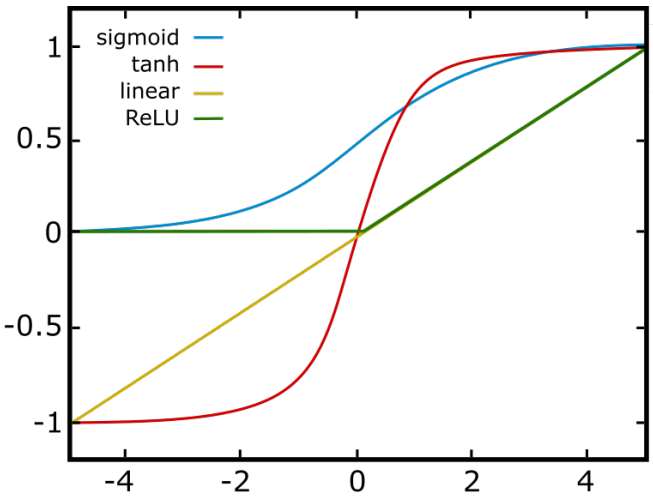


Fig. 2.2: Example of different activation functions.

It is hence linear with x whenever this is positive and 0 otherwise. In other words, it is a thresholding function with threshold in 0. The use of ReLUs has been observed to be the most important factor to improve the performance of a neural network [39]. Nevertheless, also ReLUs suffer from training issues when deployed in deep NNs, therefore further solutions were sought and this brought to the development of generalizations of the ReLU function (e.g. leakyReLU[40], PReLU [41], ELU[42], Swish [43], and so on) which often proved to be more effective than the original form [44]. Fig. 2.2 shows some of these for comparison. More in general, the literature is rich with examples regarding how the use of activation functions has improved the performance of neural networks [45–50].

Another important algorithmic change was the introduction of Dropout [51]. The core idea behind dropout is to randomly "drop out" a subset of neurons during the training phase of a neural network. This means that for each training iteration, a portion of neurons is temporarily removed from the network, along with all their incoming and outgoing connections. By doing so, dropout prevents the network from becoming overly reliant on any particular neurons, thereby reducing the risk of overfitting.

The practical effect of dropout is akin to training a large number of different neural network architectures in parallel and then averaging their predictions during the test phase. This ensemble-like behaviour enhances the generalization capability of the model. Dropout is typically applied only during the training phase and disabled during testing, where the full network is used to make predictions. The introduction of dropout has become a standard technique in the deep learning community due to its simplicity and effectiveness in improving the

performance of neural networks across various tasks and domains, although as network architecture moves away from dense connectivity, the need for dropout also fades.

2.1.3 Cost Functions

The backpropagation algorithm represents the learning backbone of most of the conventional DL approaches. Nevertheless, what truly defines the goal of a given DL solution is the Cost function [35]. Cost functions, also known as loss functions, play a pivotal role in training neural networks. Cost functions are crucial because they provide a measure of how well the neural network is performing. As outlined in the previous section, during training, the backpropagation algorithm uses the gradient of the cost function with respect to the network's weights to update them in a way that reduces the cost. This iterative process, driven by the cost function, helps the model learn from the data and improve its predictions.

For a cost function to be effective in a deep learning context, it must have the following features:

- **Differentiability.** The cost function must be differentiable with respect to the model parameters (weights and biases). This is essential because backpropagation relies on computing gradients to update the parameters.
- **Convexity (Desirable but not Necessary).** While not strictly required, convex cost functions are easier to optimize because they have a single global minimum. However, many deep learning models use non-convex cost functions due to the complexity of the networks, and optimization techniques are designed to handle such scenarios.
- **Continuity.** The cost function should be continuous to ensure smooth updates during the training process. Discontinuities can lead to unstable training and convergence issues.
- **Representation of Error.** The cost function should accurately reflect the discrepancy between the predicted and actual values. It should penalize larger errors more heavily to encourage the model to reduce significant discrepancies.

Several cost functions are commonly used in deep learning, depending on the nature of the problem (regression, classification, etc.). One example is the Mean Squared Error (MSE). This is commonly used in regression tasks as it measures

the average squared difference between the predicted and actual values. It is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \tag{2.6}$$

where $\hat{y}_i$ is the predicted value, y_i is the actual value, and n is the number of samples.

Another example is given by the Cross-Entropy loss function. Commonly used for classification tasks, the cross-entropy loss measures the difference between the true probability distribution and the predicted probability distribution. For binary classification, it is defined as:

$$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{2.7}$$

For multi-class classification, the categorical cross-entropy loss is used:

$$\text{Categorical Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}\log(\hat{y}_{ij}) \tag{2.8}$$

where c is the number of classes, y_{ij} is a binary indicator (0 or 1) if class label j is the correct classification for sample i, and $\hat{y}_{ij}$ is the predicted probability of sample i being in class j.

The above represent two of the most commonly used cost functions, however, the literature is rich with a plethora of different ones for various tasks [52]. In this work, where backpropagation-based algorithms are employed, loss function selection plays a pivotal role in the success of a learning pipeline, so much so that ad-hoc functions are devised to improve the performance of a system on specific tasks.

2.1.4 The Backpropagation Algorithm

In the preceding sections, it was hinted how Backpropagation has been a cornerstone algorithm in the training of artificial neural networks. The backpropagation algorithm is a supervised learning technique used for minimizing the error in the network's predictions. It gained widespread recognition in the 1980s, with the seminal paper by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams [33], and became the standard method for training neural networks. It builds on the simple idea that for a system to reach some (local) minimum, parameters can be adjusted in a direction that is opposite to the derivative (or gradient

with respect to a variable in case of multi-variate functions) of the function that needs to be minimized. The gradient of a function represents the direction and rate of the steepest increase in the function's value, so moving in the opposite direction (the negative gradient) leads to the steepest decrease, thus minimizing the function. This concept is also known as the Steepest Descent or Gradient Descent (GD) and is at the core of the Backpropagation algorithm.

Backpropagation operates through a two-phase process: the forward pass and the backward pass. In the forward pass, the input data is fed through the network to produce an output. Let's denote the input vector as $\mathbf{x}$. Each layer of the network consists of weights ($\mathbf{W}$), biases ($\mathbf{b}$), and activation functions (σ).

For a single neuron in a layer, the output a is calculated as:

$$a = \sigma(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}). \tag{2.9}$$

This process is repeated for each layer until the final output is produced.

The backward pass involves computing the gradient of the loss function with respect to each weight and bias in the network and then updating these parameters to minimize the loss. This involves the use of the chain rule from calculus. The chain rule helps in calculating the derivatives of composite functions. In the context of backpropagation, if we want to find the gradient of the loss L with respect to the weights $\mathbf{W}$, we need to consider how changes in $\mathbf{W}$ affect the loss through each intermediate variable.

Suppose the loss L depends on an intermediate variable z, which in turn depends on y, and y depends on x:

$$L = L(z), \quad z = z(y), \quad y = y(x) \tag{2.10}$$

Using the chain rule, the derivative of L with respect to x is:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.11}$$

In neural networks, this means the gradient of the loss with respect to the weights is computed layer by layer, starting from the output layer and moving backward to the input layer.

GD is used to update the weights and biases to minimize the loss function. If θ represents the weights and biases, the update rule is:

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta} \tag{2.12}$$

where:

- η is the learning rate,
- $\frac{\partial L}{\partial \theta}$ is the gradient of the loss with respect to the parameters.

The use of the learning rate serves to perform small enough steps to not overshoot beyond the minima, however, using too small values can reduce convergence speed or hamper reaching minima altogether.

By iteratively performing these steps, the network learns to minimize the loss function, effectively training the model.

2.1.5 Optimizers

Backpropagation refers to the methodology used to compute gradients across a neural network, facilitating the optimization process. These gradients are subsequently utilized to update the network's weights, which is achieved through the application of the GD algorithm, as introduced earlier. GD is an iterative optimization algorithm, or optimizer, designed to identify minima, ideally global minima, within a given loss landscape. In the context of deep learning, backpropagation and GD are intrinsically linked, as the latter relies on the former to compute the necessary gradients for optimization.

GD operates by leveraging the entire dataset to compute updates, and, with an appropriately chosen learning rate, it converges gradually but steadily toward the minima. However, the application of GD to large datasets poses significant challenges. The requirement to process the entire dataset for each update results in substantial computational and memory overhead, often rendering the approach impractical for modern deep learning tasks. Moreover, the lack of stochasticity in GD, owing to its use of the complete dataset, can cause the algorithm to converge to suboptimal local minima rather than global ones [35].

To address the limitations of GD, Stochastic Gradient Descent (SGD) is commonly employed as an alternative. Unlike GD, SGD updates weights by computing gradients using a single randomly selected data point or a small subset of data points (mini-batch). This approach reduces the computational burden, making optimization feasible even for large datasets. Additionally, the inherent noise introduced by using only a subset of data during each update aids in escaping local minima, thus increasing the likelihood of finding better solutions.

However, the adoption of SGD introduces new challenges, such as the need for careful fine-tuning of hyperparameters, including the learning rate and mini-batch size. While SGD often achieves faster initial progress compared to GD,

it is also more susceptible to divergence, where the optimization process fails to converge and the performance deteriorates. Proper tuning of hyperparameters is therefore critical to ensure that SGD achieves convergence to meaningful minima points and maintains stable learning dynamics.

Stochastic Gradient Descent (SGD) remains one of the most widely utilized optimization algorithms in deep learning. However, over the years, more sophisticated variants have been developed to address its limitations and improve convergence. One such variant which is frequently employed is the Adaptive Moment Estimation (Adam) algorithm. This algorithm is particularly notable for its use of the first and second moments of the loss function to accelerate convergence and its ability to define parameter-specific adaptive learning rates. The Adam algorithm is employed in this thesis in Chapters 5 and 6.

The Adam algorithm is mathematically defined by the following set of equations:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \frac{\partial L}{\partial \theta}, \tag{2.13}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \frac{\partial^2 L}{\partial \theta^2}, \tag{2.14}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.15}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{2.16}$$

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.17}$$

where m_t and v_t identify the first and second-order moments of the gradient of L with respect to θ respectively, $\hat{m}_t$ and $\hat{v}_t$ are the initialization bias corrections (to account for the moments being initialized to 0), and θ_t is the set of weights to be updated. β_1 and β_2 are two hyper-parameters that regulate the effect of the first and second-order moments on the overall weight update, η is the learning rate and ϵ is a small constant to avoid division by 0.

In general terms, all optimizers require the definition of some hyperparameters. While some of them are optimizer-specific, the learning rate and the (mini-) batch size are necessary. The learning rate regulates the extent of each weight update. As seen in the previous section, backpropagation allows the calculation of the error through different layers in a neural network. The weight update is then calculated to be inversely proportional to the gradient of the cost function. The magnitude of such updates, however, requires regulation. As a matter of

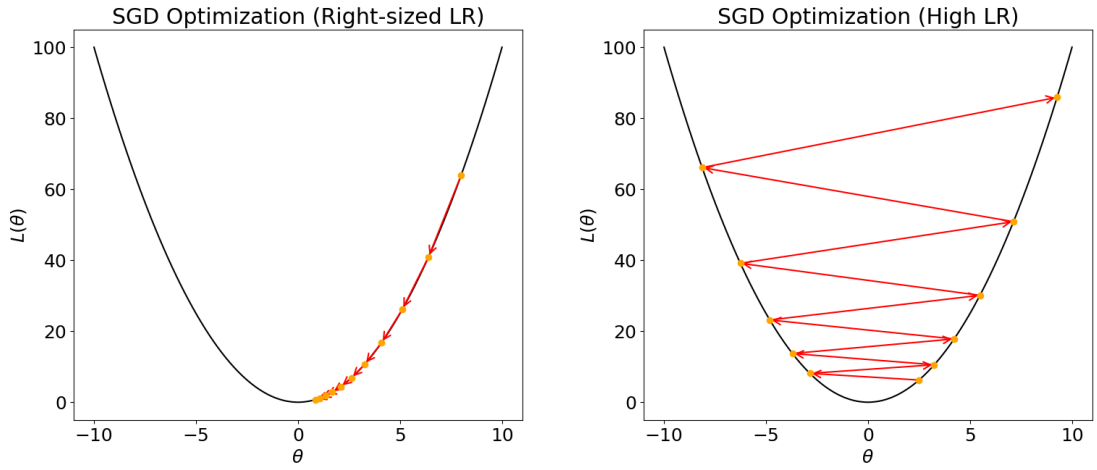


Fig. 2.3: Examples of SGD optimization with right-sized (left) and high LR (right). The red arrows represent the direction of the next update as determined by the algorithm. The plots show how, despite the right-hand side example starting from a lower loss value, the updates to the parameters θ lead to a diverging behaviour, ultimately reaching a higher loss value.

fact, if the weight update is too large the optimization process can end up diverging from the minima instead of converging to them. Figure 2.3 depicts this possibility. In the left-hand side plot, thanks to the use of a learning rate that scales down the weight updates, the algorithm converges steadily to the minimum point. In the right-hand side plot, while approaching the local minima, the weight updates drive the optimization to constantly overshoot the minimum, effectively diverging from it. The batch size, on the other hand, regulates how much of the data is processed at one time, and how much of it is considered per each optimization step. Using a batch size that equals the size of the dataset degenerates SGD into GD. Using a batch size of one allows for more frequent weight updates and lower memory utilization, but can lead to slow convergence and decreased generalization ability. The right value to use for the batch size is thus a delicate and task-specific process and a correct sizing can make the difference between a solution converging to some minima and one not. In this work, the batch size varies depending on the task at hand and on the hardware constraints.

2.2 Neuromorphic Computing

Neuromorphic Computing, similarly to the neural networks, is rooted in the integration of neural circuits into electronic systems. Both disciplines have a shared and intricate history, evolving from foundational works such as the McCulloch-Pitts Neuron [26], the perceptron [32], and the Neocognitron [53] (which models the retina). The field of NM engineering began with research into the Very-Large-Scale Integration (VLSI) of transistors, focusing particularly on their non-linear properties. Carver Mead introduced the term Neuromorphic Engineering to characterize a new engineering discipline inspired by biological design principles and architectures [54].

In modern times, NM Computing and Engineering is regarded to as an interdisciplinary subject that encompasses concepts from biology, neuroscience, mathematics, computer science, and engineering to design artificial neural systems [10]. The goal of such neural systems can be very diverse, ranging from closely replicating biological systems to providing low-power, low-latency and highly adaptable automated systems for tasks like visual information processing, audio classification and SLAM tasks, to name a few. Efforts to closely replicate biological systems include Boahen's neuromorphic circuit at Stanford University and its Braindrop (formerly Neurogrid) processor [55], Izhikevich's mathematical model of spiking neurons [56], and Eliasmith's large-scale brain modelling [57]. On the other hand, works based on the methodologies discussed in Section 2.1 merely seek to find algorithmic solutions to solve a plethora of different problems, and are only loosely inspired by the human brain. If we consider the above as the two extremes of biological plausibility and implausibility, NM computing-based solutions sit in a middle ground and try to draw the most out of the two. Notable examples include converting Neural Networks to SNNs [58–62], and employing deep learning and NNs in Neuromorphic Sensors for applications in classification [63, 64], image recognition [65, 66], and optical flow [67, 68], where Spiking Convolutional Neural Networks (SCNNs), a type of NM CNN where activation functions are essentially substituted by spiking neurons, are trained to solve these tasks. These examples underscore the increasing interest in this interdisciplinary space, allowing the advantages of NM computing and engineering, such as energy efficiency, to be combined with the ease of training provided by deep learning techniques.

This Section functions as a review of the key elements relevant to this thesis that constitute the field of NM Computing and Engineering. Section 2.2.1 delves

into the variety of spiking neuron models that have been theorized in the literature and that are often employed in developing SNNs. Section 2.2.2 introduces the concept of neural information coding and encoding systems. Section 2.2.3 details the most prominent learning methodologies for SNNs. Section 2.2.5 reports a thorough review of the available software frameworks for the development of SNNs.

2.2.1 Spiking Neuron Models

Neurons in the human brain are extremely complex cells and are present in a wide variety of functions, shapes and components [12]. Figure 2.4 shows a representation of a so-called pyramidal neuron. In here it is notable that the neuron features a specific shape, pyramidal, and has several components. The central part is called the Soma, and is were the majority of the activity is conveyed. The shorter fibres connecting into the soma are called dendrites. These are the input means to the neurons and subdivide in many smaller branches depending on the number of incoming connections that the neuron developed. The longer fibre is called the axon and is the output means of the neuron. Spikes travel through the axon. At the points of contact between one neuron's axon and another neuron's dendrites are the synapses. This is where information is passed from one into the other depending on the strength of said synapse. Furthermore, axons and dendrites are further composed of smaller blocks. Due to this level of complexity and to the inherent difficulty in studying them, researchers have often developed mathematical models that could approximate (some of) the observed behaviours. Models that approximate the compartmentalization of the neurons and their shape also exist, but they are more often employed for single-neuron analysis. Instead, this thesis focuses on the use of so-called point neurons. While this is a simplification, it can be argued that the use of a network of point neurons can effectively approximate the behaviour of a single compartment neuron, while at the same time allowing the definition of larger networks. The point Spiking Neuron Model (SNM) that have been theorized in the literature vary in their objective and their mathematical complexity. Their objective is to describe the observed behaviours of human brain neurons in response to certain stimuli [12]. A possible way to quantify their complexity is by estimating the Floating Point Operations per second (FLOPs) that are required to run the model on hardware. Tab. 2.1 is a collection of a few of them, reported with their mathematical model, typology and number of floating point operations per millisecond (FLOPS/ms) when available.. SNMs are normally characterized by a system of differential

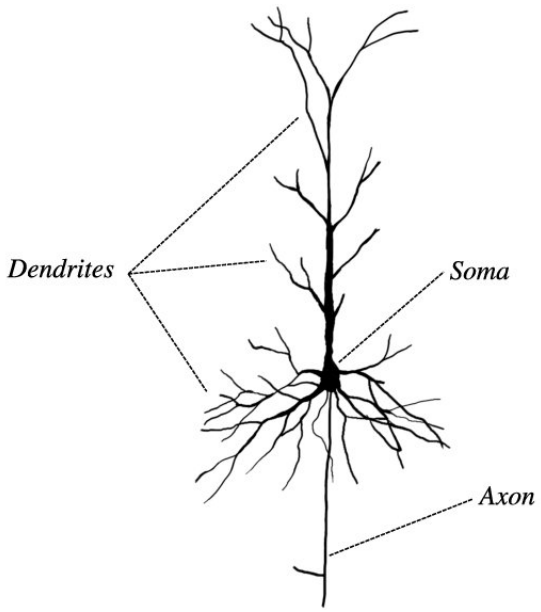


Fig. 2.4: Example of a neuron cell, adapted from [12].

equations where variables evolve through time. When one of these variables, the membrane potential, reaches a pre-defined threshold V_{th}, the spiking neuron is said to emit a spike and is reset to a certain value V_{reset}. Spikes are stereotypical signals that neurons use to communicate information and are completely characterized by the time at which they occurred (meaning that the shape of a spike is always the same, hence it has no meaning).

While conventional DL also borrows the idea of neurons, the spiking neurons used in NM computing have two crucial differences. The first one is in their output which, as mentioned earlier is, normally, a binary 0/1 spike. This allows for a network of spiking neurons to significantly reduce the amount of information propagated in the network. The second one is in the fact that they retain their state (the membrane potential) for future computations. This allows to consider the concept of sequentiality between two or more inputs. Indeed, it means that the output at time $t_1 = t_0 + \Delta t$ is influenced by the amount of potential accumulated since time t_0. Furthermore, the membrane potential of spiking neurons tends to return to a resting state over some time according to the dynamics specified in their differential equation. This further expands the concept of sequentiality to include that of time. Since the potential can vary in different ways depending on the neuron model, two or more inputs can be temporally related depending on when and on how distant in time they appeared.

The literature is rich with a plethora of different neuron models, mostly describing the dynamics of the soma (core) of the cortical neurons. These can be

roughly subdivided into two larger groups, the bio-physical or conductance-based models and the event-based or integrate-and-fire models. Whilst some of the theorized models feature an extremely fine-grain level of detail of how a brain neuron behaves, it is important to note that when transposed into simulations on software, inherent limitations are present from the need to discretize continuous variables. As such, approximation errors can incur as well as numerical instabilities in the results due to some of the differential equations needing to be resolved numerically rather than mathematically. Such limitations need to be taken into account whenever creating software simulations.

Tab. 2.1. Collection of Spiking Neuron Models. On each line, the reference, mathematical model, typology and number of FLOPS/ms are reported for each neuron model. Types of reported neurons are phenomenological (Ph), conductance-based (C) and population (P). Where no FLOPs analysis was found in the literature, N/A is inserted standing for Not Available. Spiking patterns are grouped into the three macro categories Tonic (T), Adapting (A) and Bursting (B) similarly to [12] for readability purposes.

Neuron Model	**Year**	**Mathematical Model**	**Type**	**# FLOPs/ms**	**Spiking Patterns**
Integrate&Fire (IF) [69, 70]	1907	$C \cdot \frac{du}{dt} = I$	Ph	N/A	T
IF with Adaptation[56, 71]	2003	$\begin{cases} u' = I + a - bu + g(d - u) \\ g' = \frac{(e\delta(t) - g)}{\tau} \end{cases}$	Ph	10	T, A
IF or Burst[72]	2000	$\begin{cases} u' = I + a - bu + gH(u - u_h)h(u_T - u) \\ u \leftarrow u_{reset} \quad \text{if } u = u_{thresh} \\ h' = \begin{cases} \frac{-h}{\tau^-} & \text{if } u > u_h \\ \frac{1-h}{\tau^+} & \text{if } u < u_h \end{cases} \end{cases}$	Ph	13	T, B
LeakyIF (LIF) [69, 70]	1907	$\begin{cases} \tau_m \frac{du}{dt} = u_{rest} - u(t) + RI(t) \\ u \leftarrow u_{reset} \quad \text{if } u \geq u_{thresh} \end{cases}$	Ph	5	T
Fractional Order LIF[73]	2015	$C \cdot \frac{d^\alpha u}{dt^\alpha} = -g_L(u - u_L) + I$	Ph	N/A	T, A, B

Table 2.1 continued from previous page

Neuron Model	**Year**	**Mathematical Model**	**Type**	**# FLOPs/ms**	**Spiking Patterns**
Probabilistc LIF (pLIF) [74]	2011	$\begin{cases} \tau_m \frac{du}{dt} = u_{rest} - u(t) + RI(t) & \text{LIF eq.} \\ \text{Probabilistic modulating eq.} & \text{Choose from 4} \end{cases}$	Ph	N/A	T
Resonate and Fire (IZ) [75]	2001	$\begin{cases} z' = I + (b + i\omega)z \\ z \leftarrow z_0(z) & \text{if } Im\, z = a_{thresh} \end{cases}$	Ph	10	T, B
Resonate and Fire (Frady) [19, 76]	2019	$z_k[t] = \lambda_k e^{i\omega\Delta t} z_k[t-1] + a_k[t]$ $\phi[t] = \begin{cases} \Re(z_k[t]) & \text{if } \begin{cases} \Im(z_k[t]) = 0 \\ \Re(z_k[t]) >= V_{th} \end{cases} \\ 0 & \text{otherwise} \end{cases},$	Ph	N/A	T, B
Exponential IF (EIF) [77]	2003	$\begin{cases} \tau_m \frac{du}{dt} = -(u(t) - u_{rest}) + \Delta_T \exp\left(\frac{u(t) - \Theta_{rh}}{\Delta_T}\right) + R \cdot I \\ u \leftarrow u_{reset} & u = u_{thresh} \end{cases}$	Ph	N/A	T, B
Adaptive EIF (AdEx) [78]	2005	$\begin{cases} \tau_m \frac{du}{dt} = -(u - u_{\text{rest}}) + \Delta_T \exp\left(\frac{u - \vartheta_{\text{rh}}}{\Delta_T}\right) - Rw + RI(t), \\ \tau_w \frac{dw}{dt} = a(u - u_{\text{rest}}) - w + b_w \sum_{t^{(f)}} \delta(t - t^{(f)}) \end{cases}$	Ph	N/A	T, A, B

Table 2.1 continued from previous page

Neuron Model	**Year**	**Mathematical Model**	**Type**	**# FLOPs/ms**	**Spiking Patterns**
QuadraticIF (QIF) or Theta Neuron [79]	1986	$\begin{cases} u' = I + a(u - u_{rest})(u - u_{thresh}) & \\ u \leftarrow u_{reset} & u = u_{peak} \end{cases}$	Ph	7	T
Izhikevich's Neuron[16]	2003	$\begin{cases} u' = 0.04u^2 + 5u + 140 - v + I & \\ v' = a(bu - v) & \\ u \leftarrow c & \text{if } u \geq u_{thresh} \\ v \leftarrow v + d & \text{if } u \geq u_{thresh} \end{cases}$	Ph	13	T, A, B
Quartic Neuron [80]	2008	$\begin{cases} u' = F(u) - w + I \\ w' = a(b \cdot u - w) \\ F(u) = u^4 + 2a \cdot u \end{cases}$	Ph	N/A	T, A, B
Spike Response Model (SRM) [12]	1993	$u(t) = \eta(t - \hat{t}) + \int_0^\infty \kappa(t - \hat{t}, s) I(t - s) ds$	Ph	N/A	T, A, B
Probabilistic Spiking Neuron Model (pSNM)[81]	2010	$PSP_i(t) = \sum_{p=t_0,\ldots,t} \sum_{j=1,\ldots,m} e_j g(p_{c_{j,i}}(t-p)) f(p_{s_{j,i}}(t-p)) w_{j,i}(t) + \eta(t - t_0)$	Ph	N/A	T

Table 2.1 continued from previous page

Neuron Model	**Year**	**Mathematical Model**	**Type**	**# FLOPs/ms**	**Spiking Patterns**
FitzHugh-Nagumo Model [82, 83]	1961	$\begin{cases} u' = a + bu + cu^2 + du^3 - v \\ v' = \epsilon(eu - v) \end{cases}$	C	72	T
Hindmarsh-Rose Model [84]	1984	$\begin{cases} u' = v - F(u) + I - \omega \\ v' = G(u) - v \\ \omega' = \frac{(H(u)-\omega)}{\tau} \end{cases}$	C	120	T, A, B
Morris-Lecar [85]	1981	$\begin{cases} C \cdot \frac{du}{dt} = -g_{Ca}M_{ss}(u)(u-u_{Ca}) - g_K w(u-u_K) - g_L(u-u_L) + I \\ \frac{dw}{dt} = \frac{w_{ss}(u)-w}{T_w(u)} \end{cases}$	C	600	T, B
Hodgkin-Huxley Neuron [86]	1952	$\begin{cases} C \cdot \frac{du}{dt} = -g_K n^4(u-u_K) - g_{Na}m^3h(u-u_{Na}) - g_l(u-u_l) + I \\ \frac{dn}{dt} = \alpha_n(u)(1-n) - \beta_n(u) \cdot n \\ \frac{dm}{dt} = \alpha_m(u)(1-m) - \beta_m(u) \cdot m \\ \frac{dh}{dt} = \alpha_h(u)(1-h) - \beta_h(u) \cdot h \end{cases}$	C	1200	T, A, B

Table 2.1 continued from previous page

Neuron Model	**Year**	**Mathematical Model**	**Type**	**# FLOPs/ms**	**Spiking Patterns**
Wilson Polynomial Neurons [56, 87, 88]	1972	$\begin{cases} \frac{dx_1}{dt} = -ax_1 + \frac{wx_1 - by_1 + \alpha_1 x_2 + I_1}{\sqrt{1+(wx_1 - by_1 + \alpha_1 x_2 + I_1)^2}} \\ \frac{dy_1}{dt} = -dy_1 + \frac{cx_1 - ey_1 + \beta_1 x_2 + J_1}{\sqrt{1+(cx_1 - ey_1 + \beta_1 x_2 + J_1)^2}} \\ \frac{dx_2}{dt} = -ax_2 + \frac{wx_2 - by_2 + \alpha_2 x_1 + I_2}{\sqrt{1+(wx_2 - by_2 + \alpha_2 x_1 + I_2)^2}} \\ \frac{dy_2}{dt} = -dy_2 + \frac{cx_2 - ey_2 + \beta_2 x_1 + J_2}{\sqrt{1+(cx_2 - ey_2 + \beta_2 x_1 + J_2)^2}} \end{cases}$	P	180	T, A, B

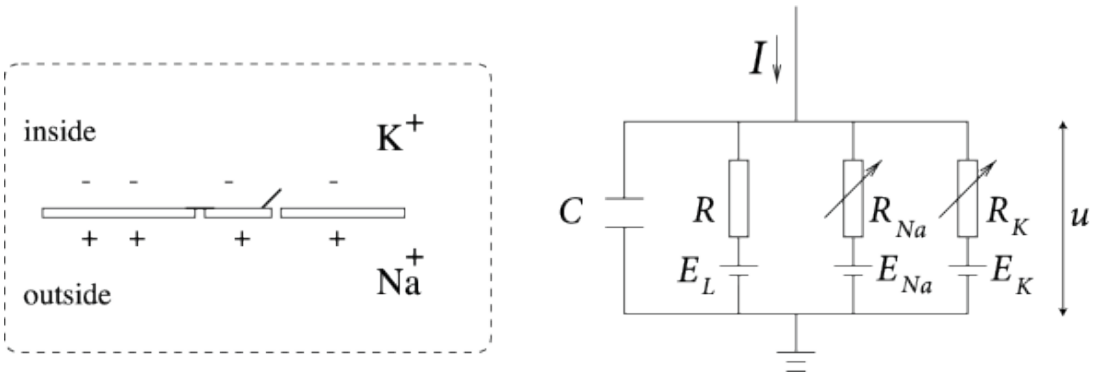


Fig. 2.5: Schematics of the Hodgkin-Huxley neuron model from the Neuronal Dynamics book [12].

2.2.1.1 Conductance-based Models

This class of neuron models is characterized by the fact that all the variables and parameters present in the model have a biophysical correspondence and are therefore measurable through experiments [89]. Among them, the Hodgkin-Huxley's (HH) model [86]is considered to be one of the most important in computational neuroscience and defines a system of 4 non-linear differential equation with four variables and a number of parameters. The schematics of the model can be found in Fig. 2.5, where its equivalent circuit is reported. While higher levels of complexity can be reached by including further variables in the model, this is not amenable to mathematical analysis. In fact, other simpler conductance-based models have been derived in the literature in order to ease the analysis while still retaining biophysical plausibility. Some examples are the FitzHugh-Nagumo model [82, 83], the Hindmarsh-Rose [84] and the Morris-Lecar model [85]. Nevertheless, they still remain rather complex for what concerns analysis and computation, therefore this family of neuron models is often used only when studying single-cell or small population dynamics [56].

2.2.1.2 Integrate-and-Fire Models

The family of Integrate-and-Fire (IF) or phenomenological neuron models comprises all those models that aim to model the spike generation mechanism, rather than the specific biophysical components of neurons [12]. Integrate-and-fire models require at least two equations, one describing the dynamics of the membrane potential and the other one defining the action potential generation. Events are integrated over time and convey electrical charges that can cause excitation or inhibition of the membrane potential of the receiving neuron. The literature has seen the definition of several different types of neurons in this family. Neverthe-

less, the most prominently used one in NM computing applications and SNNs is the Leaky Integrate-and-Fire (LIF) [61, 90–99], despite its limitations in terms of representational power (see Section 2.2.1.2.1), due to its simplicity and computational efficiency. This section covers some amongst the most relevant models that have been considered for the development of Spiking Neural Networks.

2.2.1.2.1 Leaky Integrate-and-Fire The simplest model, apart from the perfect integrator, is the Leaky Integrate-and-Fire [70]. The dynamics of the membrane potential are here described by the following linear differential equation:

$$\tau_m \frac{du}{dt} = -(u(t) - u_{rest}) + R \cdot I \tag{2.18}$$

where τ_m is the membrane time constant, $u(t)$ is the membrane potential as a function of time, u_{rest} is the resting potential of the membrane, R is a resistance and I is the incoming current.

Although it has been shown that this model lacks the ability to describe the most of the neuronal spiking patterns and sub-threshold behaviors [12, 56], it is the most common choice for the development of large scale neural networks mostly because of its efficiency [61, 90–99]. By assuming that at time $t = t_0$ the membrane potential takes the value $u(t_0) = u_{rest} + \Delta u$, and that for $t > t_0$, the input current becomes null $I = 0$, the solution to (2.18) would be the following:

$$u(t) = u_{rest} + \Delta u \exp\left(-\frac{t - t_0}{\tau_m}\right) \quad \text{for } t > t_0. \tag{2.19}$$

Hence the membrane potential would decay at a pace dependent on τ_m until returning to the resting potential u_{rest}. As mentioned earlier, spiking neurons communicate information through short pulses called spikes. It is thus possible to write the current as an amount of charge q being carried in a very short moment in time modelled by a Dirac δ-function, $I = q\delta(t)$. By doing this, when a neuron in it's resting state $u(t_0) = u_{rest}$ receives such a pulse current, the membrane potential will make a jump $\Delta u = \frac{q}{C}$ [12] and will then continue evolving according to (2.19). While the above is a correct formulation of the LIF neuron model in a continuous time domain, when developing software implementations the time is commonly discretized into time steps [100]. As such, different formulations of the LIF neuron model can be found that approximate in a way or an other the above. A common discrete formulation considers a decay or leak factor $\alpha \in [0, 1]$

that retains a percentage of the membrane potential at the previous time step:

$$u[t] = (1 - \alpha)u[t - 1] + x[t], \tag{2.20}$$

where $u[t]$ represent the membrane potential at discrete time t, α is the decay constant and $x[t]$ is the input current. The state of the neuron is thus calculated at every time step. A smaller time step will thus result in more precise calculations, but longer simulation times [100], and viceversa. Similar discretization strategies are also considered for other SNMs.

2.2.1.2.2 Exponential Integrate-and-Fire The Exponential Integrate-and-Fire (EIF) [77] expands on the LIF neuron by adding an exponential dependency from the previous state of the membrane potential in an attempt to account for some non-linear dynamics witnessed from observations [12]. The dynamics of the membrane potential are described by the following differential equation:

$$\tau_m \frac{du}{dt} = -(u(t) - u_{rest}) + \Delta_T \exp\left(\frac{u(t) - \Theta_{rh}}{\Delta_T}\right) + R \cdot I, \tag{2.21}$$

where Δ_T is a parameter determining the sharpness of the exponential curve and Θ_{rh} is the rheobase threshold. When $u > \Theta_{rh}$ the exponential term becomes prominent over the linear one, leading to an upswing of the curve that takes the membrane potential to infinity in finite time. Smaller values of Δ_T make the upswing extremely sharp, and in the limit of $\Delta_T \to 0$ the EIF becomes a LIF model with a firing threshold in $V_{th} = \Theta_{rh}$. A visual representation of the patterns above can be found in Fig. 2.6.

2.2.1.2.3 Quadratic Integrate-and-Fire The Quadratic Integrate-and-Fire (QIF) or Theta neuron [79, 89], employs a quadratic dependency from the previous state of the membrane potential:

$$\tau_m \frac{du}{dt} = a_0(u(t) - u_c)(u(t) - u_{rest}) + R \cdot I, \tag{2.22}$$

where a_0 is a parameter of the model that regulates the magnitude of the dependency from the membrane potential and u_c is a cut-off threshold such that when $I = 0$ and $u > u_c$ the membrane potential grows until the emission of a spike. Typical values of a_0 are in the order of 10^{-2} and it affects the quadratic curve by making it sharper and very negative for bigger values and smoother and less negative for smaller ones.

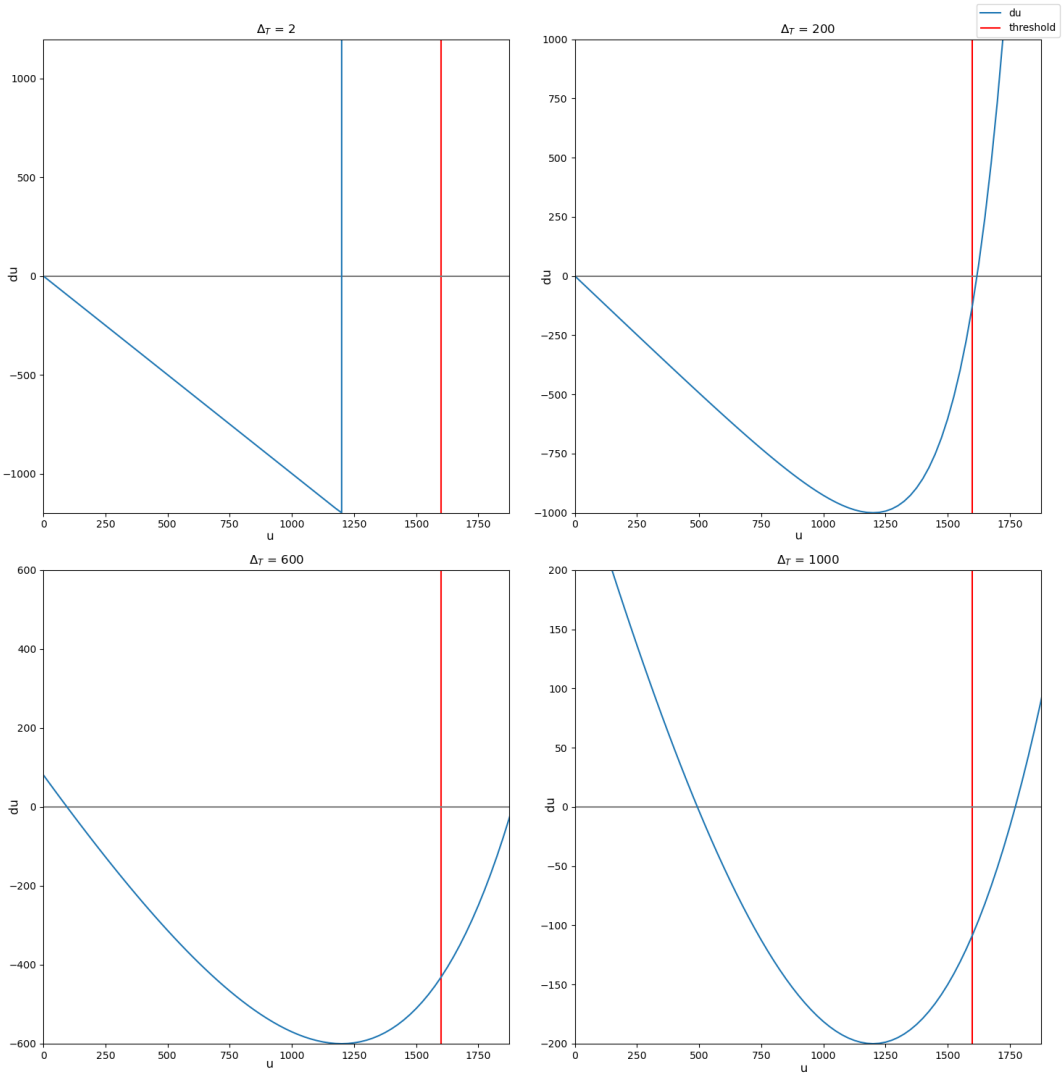


Fig. 2.6: Example of EIF neurons with fixed Θ_{rh} and varying Δ_T. On the x-axis the membrane potential u, on the y-axis its next state update du. In the top left corner the curve approximates a LIF.

2.2.1.2.4 Resonate-and-Fire Resonate-and-Fire (RF) neurons are a particular type of phenomenological neurons whose internal state is defined by a complex number. The first model of RF neuron was theorized by Izhikevich in 2001 [75] when he presented a complex-valued state neuron that showed oscillating sub-threshold behaviours when excited. Such model is described by the following formulation:

$$\begin{cases} z' = I + (b + i\omega)z \\ z \leftarrow z_0(z) \qquad\qquad \text{if } Im\, z = a_{thresh}, \end{cases} \tag{2.23}$$

where I is the input current, $(b + i\omega)$ is an internal parameter, z is the neuron's complex state, z_0 is the reset value or function, and a_{thresh} is the threshold to be reached for spike emission. For an RF neuron to emit spikes, the incoming stimulation is required to be nearly resonant with the neuron's internal oscillation frequency. More relevant to the work presented in this thesis however, is the RF neuron version proposed by Frady et al. [19, 76]. Their proposed model slightly differs from Izhikevich's one and can be formulated in the following way. Let $z_k = a + ib$ be the internal complex-valued state of the k-th RF neuron. Then the update rule for the state of the RF neuron at time step t can be expressed as:

$$\begin{aligned} z_k[t] &= \lambda_k e^{i\omega\Delta t} z_k[t-1] + a_k[t] \\ \phi[t] &= \begin{cases} \Re(z_k[t]) & \text{if } \begin{cases} \Im(z_k[t]) = 0 \\ \Re(z_k[t]) >= V_{th} \end{cases} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2.24}$$

Here, λ_k is the decay factor, ω is the resonant frequency, Δt is the timestep duration, and $a_k[t]$ represents the accumulated synaptic input at timestep t. $\phi[t]$ represents the spiking function, which produces a (graded) spike equal to the neuron's state Real part $\Re(z_k[t])$ whenever the real part is above a set threshold V_{th} and the imaginary part $\Im(z_k[t])$ crosses the 0.

By comparing this with the previous model, the following points can be highlighted:

- Both models are governed by their resonant frequency ω, with the one by Frady et al. explicitating a decay factor $\lambda_k \in (0, 1)$.
- The latter model features a different spiking function that considers both the imaginary and real part of the neuron's state and emits spikes with value $\neq 0$, thus allowing implementation only in modern NM chips [101].

- The latter model does not have a reset function when a spike is emitted.

This type of RF neuron can be used to perform efficient spectral analysis by acting as a bank of filters that compute the Short-Time Fourier Transform (STFT) of input signals. The neurons output complex-valued coefficients as a timing pattern of graded spikes, encoding the spectral information in a sparse and event-driven manner. This significantly reduces the communication bandwidth compared to conventional STFT methods while maintaining high reconstruction accuracy.

2.2.1.2.5 Adaptive Spiking Neurons Both the QIF and the EIF feature a further level of complexity with respect to the LIF due to the inclusion of non-linear dependencies that affect both the computational costs and the ease of analysis but allow for a more precise generation of spikes [12]. Additionally, a hidden cost lies in the use of two extra parameters in each of them.

With the inclusion of adaptation variables within the neuron model, it is possible to account for a yet larger number of spiking patterns and to render possible the manifestation of spike bursts, spike-adaptation responses and irregular spiking [12]. This comes at the cost of more differential equations in the model (one per variable) and two relevant examples are given by the Adaptive Exponential Integrate-and-Fire (AdEx) [78] neuron model which builds on top of the EIF, and by Izhikevich's (IZ) neuron model [16] which builds on top of the QIF. Their mathematical model can be found in Tab. 2.1. Another example that is relevant to this thesis is given by the Current-Based Leaky Integrate-and-Fire (CuBa) [102] neuron model where the dendritic current is modulated by a separate differential equation as described in the following:

$$\begin{aligned} u[t] &= (1-\alpha_u)\, u[t-1] + x[t] \\ v[t] &= (1-\alpha_v)\, v[t-1] + u[t] + \text{bias} \\ s[t] &= v[t] \geq \vartheta \\ v[t] &= v[t]\,(1-s[t]), \end{aligned} \tag{2.25}$$

where $u[t]$ represents the dendritic current at time t, $x[t]$ is the input to the dendrites, α_u is the current's decay rate, α_v is the voltage's decay rate, $v[t]$ is the neuron's internal voltage at time t, and ϑ is a set threshold. The presence of the dendritic current with a programmable decay allows for a wider range of applications and for the retention, to some extent, of previous inputs in a way similar to a recursive dendritic connection. When $\alpha_u = 1$, the CuBa model becomes a standard LIF neuron.

A number of neuron models have been theorized in the literature, all answering different modelling needs or considering different aspects of the observed neuronal behaviours. The ones cited above are amongst the most relevant for what concerns this thesis and NM computing. As reported above, the LIF model is the most widely used in the development of SNN for NM applications, but at the same time, models like the AdEx and Izhikevich's have received a lot of attention in the literature. The EIF and the QIF are on one hand the baseline of the AdEx and IZ models respectively, and, on the other hand, a slightly more complex single-variable alternative to the LIF neuron model. The CuBa neuron model represents a generalized version of the LIF with the capacity to retain incoming information. The RF neuron represents yet another typology of neuron that can be exploited for engineering applications thanks to its features.

2.2.1.3 Analyses of Spiking Neurons in the Literature

Although most of the works relating to using SNNs for ML applications focus on employing a type of neuron without concern for their representational abilities (see Section 2.2.1.2), the literature reports of some works analysing the neuron models more specifically. Many of these works on spiking neurons concentrate on the neurobiological aspects they expose. One of the most influential works in this matter is the one by Izhikevich [56], which compared several models of spiking neurons (LIF, LIF with adaptation, LIF-or-burst, resonate-and-fire, QIF, Izhikevich's, FitzHugh-Nagumo, Hindmarsh-Rose, Morris-Lecar, Wilson, Hodgkin-Huxley), outlining their ability to reproduce observed neuronal behaviours and the cost (in terms of floating-point operations) of implementing such neurons in software applications. Similar work was conducted in [103], but focusing on a smaller subset of neurons (LIF, Izhikevich's, FitzHugh-Nagumo, Wilson, Hodgkin-Huxley) and analysing their numerical stability. Although they closely study spiking neuron models, the two studies above concentrate on their computational costs and the intrinsic biological mechanics that each model can reproduce. However, they do not consider the effect of using spiking neurons with different dynamics in an ML system. For example, they do not test whether different types of neuron models enable learning different features, or achieving higher performance levels in certain tasks.
A number of other works concentrate on the efficacy of neuron models in representing observed cortical neuron firing patterns. One example is given by [104], where the authors make an exploratory analysis of how parameters influence

Izhikevich's neurons in showing different spiking patterns. Still regarding Izhikevich's neurons, Kumar et al. [105] estimate parameters that allow the neuron model to optimally reproduce a given spike train. Teeter et al. [106] use a generalized version of the LIF neuron model (GLIF) to understand whether more complex models allow predicting spike timing behaviours more closely; they conclude that this ability does not increase monotonically with the complexity, nor with the ability to reproduce sub-threshold dynamics. In [107] it is argued that integrate-and-fire neuron models are good enough estimators of input spike trains when coupled with an adaptation variable. This is both quantitatively and qualitatively shown by the authors and provides a solid ground for the adoption of this family of neuron models.
Finally, in [108] the authors compare different neuron models embedded in a liquid state machine (LSM), a particular type of reservoir computing network. They evaluate their model with two different input patterns and use Euclidean distance and entropy to estimate the "separation" ability of the LSM, i.e. its ability to generate different response patterns for different stimuli. They test their LSM using six spiking neuron models (IF, resonate-and-fire, FitzHugh-Nagumo, Hindmarsh-Rose, Morris-Lecar, and Izhikevich's) and perform experiments varying the density of the connections between the neurons. They found that the LSM failed to achieve satisfactory levels of separation only when using Izhikevich's neurons. Other models allowed better separation levels depending on the density of the connections. They conclude by postulating that, for LSM implementations, Morris-Lecar, resonate-and-fire, and Hindmarsh-Rose models are most suitable.
Overall, the works reviewed above offer different kinds of insights into the use of spiking neuron models. They demonstrate that differences are in fact found in the way they produce spike patterns, or that the inclusion of complexity per se is not necessarily going to improve the model's performance. Nevertheless, with a few exceptions, the focus of these studies often relates to neurobiological investigations, rather than ML or engineering ones. NM computing and SNNs are more often being employed to approach tasks that relate to everyday uses, like classification, tracking and forecasting [10]. They are thus a function by means of which it is possible to resolve engineering problems. As such, investigating the employment of different types of spiking neurons as a way to improve an SNN's performance on specific tasks could be an interesting research development direction.

2.2.2 Neural Coding Algorithms

In order to understand what the spikes exchanged within an SNN signify, the concept of neural coding needs to be introduced. This is about determining what conveys information in spike trains, however, the exact mechanisms of spiking neuron coding remain a debated topic in neuroscience [109–111]. Neural coding implementations generally fall into two categories: rate-based and temporal-based. Fig. 2.7 presents popular implementations from both categories, which are elaborated upon in the following sections.

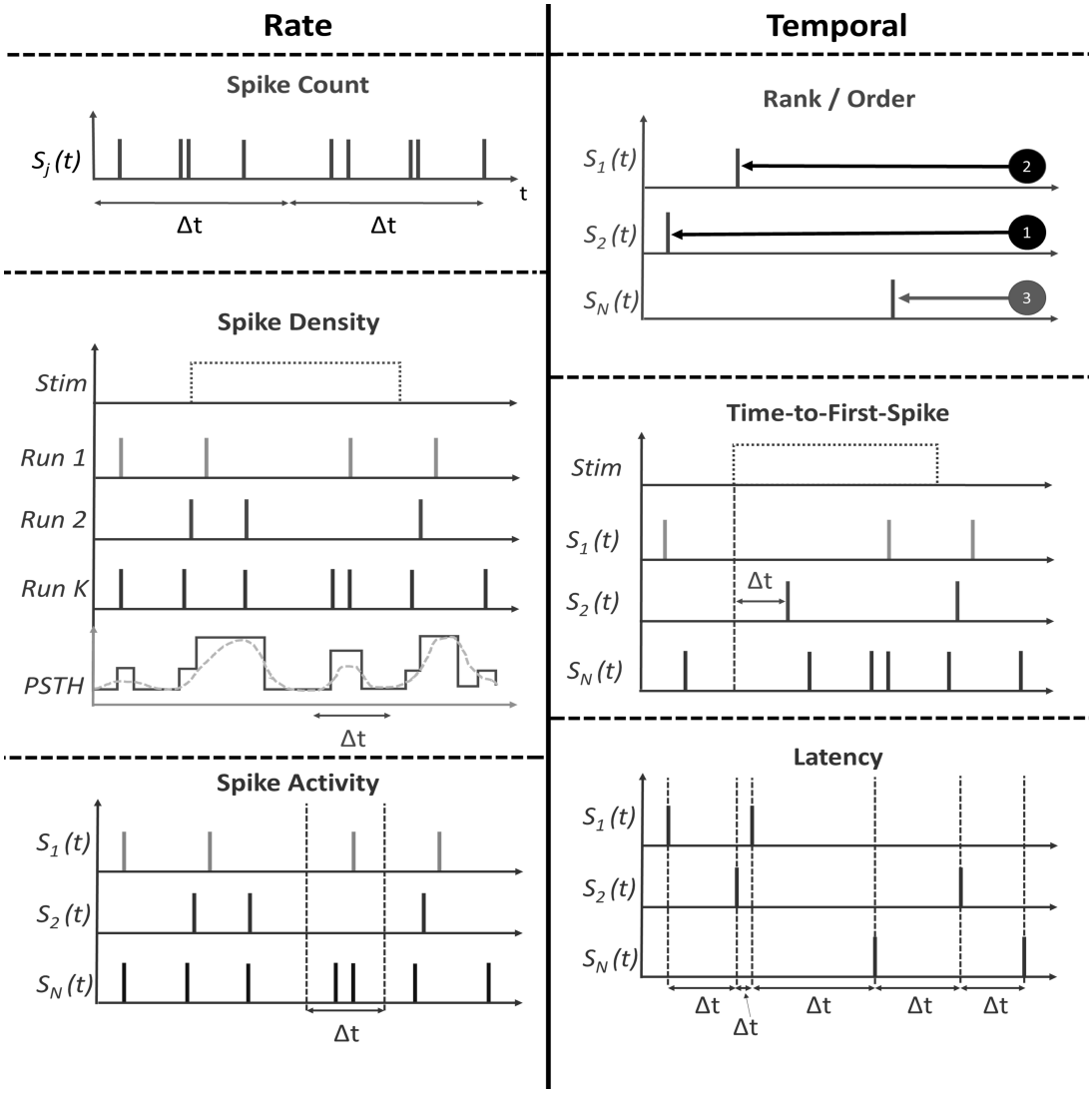


Fig. 2.7: Illustration of three popular coding schemes for rate-based and temporal-based neuron coding.

2.2.2.1 Rate Coding

Rate coding refers to methods that use firing rates for encoding information. This means that in a spike train, the information is conveyed by the number of spikes present in a certain window of time. This section explores three common methods: Spike Counter, Spike Density, and Spiking Population Activity.

2.2.2.1.1 Spike Counter *Spike Counter* [12] considers the temporal average of spikes in a time window Δt, thus conveying information in the frequency of spikes. This is visually shown in the top left of Fig. 2.7, and can be formulated in the following way:

$$Count(t) = \frac{n_{sp(\Delta t)}}{\Delta t} \tag{2.26}$$

2.2.2.1.2 Spike Density *Spike Density* [12] is about applying the spike counter method across different stimuli to the same neuron, thus obtaining a Peri-Stimulus-Time Histogram (PSTH). The number of spike occurrences within a predefined time window over n_K runs K is summed and divided by both Δt and K to yield the spike density PSTH:

$$Density(t) = \frac{1}{\Delta t}\frac{n_K(t; t+\Delta t)}{K} \tag{2.27}$$

2.2.2.1.3 Spiking Population Activity *Spiking Population Activity* [12] is similar to Spike Density, but it measures spiking activity across multiple neurons rather than across multiple stimuli The spike counts are thus summed from N neurons over Δt, then divides by Δt and N:

$$Activity(t) = \frac{1}{\Delta t}\frac{n_{\text{act}}(t; t+\Delta t)}{N} \tag{2.28}$$

This method reduces neuron variability and allows simultaneous representation of various stimulus attributes. Despite its advantages, it requires identical neuron connections, posing biological and implementation challenges [112, 113].

2.2.2.2 Temporal Coding

Temporal coding, or spike coding, is an alternative to rate coding. Figure 2.7 illustrates three popular temporal coding schemes: Rank/Order Coding, Time To First Spike (TTFS), and Latency Coding.

2.2.2.2.1 Rank/Order *Rank/Order* coding, shown in the top right panel of Fig. 2.7, encodes information based on the order of neurons' first spikes. This method assumes neurons rarely spike more than once per stimulus. Thorpe et al. [114] demonstrated this through a binary classification task where subjects identified animals in images displayed for 20ms, suggesting that low latency processing required sparse order-based coding. This method has shown promise in fast object classification tasks, mimicking the visual system's low latency response [115, 116].

2.2.2.2.2 Time To First Spike *Time-to-First Spike* (TTFS), illustrated in Figure 2.7, encodes information in the time between stimulus initiation and the first spike. The firing time $T_{input} - x_i c$ of an input neuron a_i encodes the magnitude of an input variable $x_i \in \mathbb{R}$, where $c > 0$ and T_{input} is the stimulus arrival time [117, 118]. This allows low latency processing of rich information, as shown in encoding touch signals from fingers [119], and has been used in recent methods to backpropagate errors from temporal differences [120, 121].

2.2.2.2.3 Latency *Latency* coding, also depicted in Figure 2.7, extends TTFS by considering the relative timing between all consecutive spikes, not just the first. Precise spike timing plays a crucial role in the nervous system and temporal-based learning rules [122, 123]. While typically used in feed-forward networks due to the challenges of maintaining precise timing in recurrent networks, latency coding has been explored within reservoir computing to retain this precision [124, 125].

2.2.3 Learning Rules

The previous sections explored various design choices for SNNs, such as neuron models and coding schemes. However, the real challenge lies in enabling these networks to learn effectively from data. Learning involves modifying synaptic weights over time, a concept that revolutionized ANNs with the advent of backpropagation (see Section 2.1.4). Unfortunately, backpropagation is not directly applicable to SNNs due to their discrete, asynchronous spikes and non-differentiable nature.

Developing robust learning algorithms for SNNs remains a complex task, influenced by their closer resemblance to biological systems.

This section discusses two main approaches to SNN learning that are relevant to this thesis: unsupervised learning methods based on Hebbian rules and spike timing, and supervised learning techniques allowing error backpropagation.

2.2.3.1 Unsupervised Learning

Unsupervised learning in SNNs draws inspiration from biological principles, where neurons learn from local activity without specific task instructions. Hebbian Learning [126], proposed by Donald Hebb, is a foundational rule that has inspired various unsupervised approaches[127–131], but the general rule is that if a presynaptic spike occurs shortly before a neuron's firing, the relevant synaptic connection would be strengthened (Long Term Potentiation – LTP). Otherwise, the same synapse would be weakened (Long Term Depression – LTD). This process is called Spike-Timing-Dependent Plasticity (STDP) and it is believed that it could be the underlying learning and information storage processing system in the brain.

In practical terms, the weight change Δw_j at synapse j depends on the timing difference between pre- and postsynaptic spikes:

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W\left(t_i^n - t_j^f\right) \tag{2.29}$$

where $W(\Delta t)$ is an STDP learning window function. A typical function is the one formulated as:

$$W(\Delta t) = \begin{cases} A^+ e^{\frac{-\Delta t}{\tau^+}} & \Delta t > 0 \\ A^- e^{\frac{\Delta t}{\tau^-}} & \Delta t < 0 \end{cases} \tag{2.30}$$

This function is derived from experimental data [132] and models [133], with parameters A^+ and A^- and time constants τ^+ and τ^- typically around 10ms. In this thesis, the STDP learning rule has played a crucial role in the study of the spiking neuron models. In particular, the STDP learning rule that is employed is an approximation found in [134] and is formulated as follows:

$$\Delta W_{i,j} = \begin{cases} A^+ \times (W_{i,j} - LB) \times (UB - W_{i,j}) & \text{if} \quad T_j \leq T_i, \\ A^- \times (W_{i,j} - LB) \times (UB - W_{i,j}) & \text{if} \quad T_j > T_i, \end{cases} \tag{2.31}$$

where $W_{i,j}$ is the weight of the synapse connecting neuron j (pre-synaptic) to neuron i (post-synaptic), LB and UB are a lower and an upper bound value respectively, T_j is the timing of the spike emitted by neuron j, T_i the timing of the spike emitted by neuron i, and A^+ and A^- are two parameters used to scale the weight update. Through this system, the weights tend to be saturated towards 0 or 1, thus providing a form of automatic regularization and avoiding problems like the divergence of the weight values. At the same time, because weights that are closer to the saturation points receive minor weight updates, this implementation of STDP can lead to dead neurons due to partial saturation: neurons are triggered enough times to start learning certain features, but eventually "lose" the race towards learning it and are left in a state they can hardly move away from. This makes this STDP rule potentially more sensitive to the order of presentation of the data, which is the true drive for the learning of such features.

2.2.3.2 Supervised Learning

Supervised learning uses labelled data to guide the learning process and has been highly successful with ANNs. Translating this success to SNNs is challenging due to the discrete nature of spikes. However, several supervised SNN learning algorithms have been developed, such as ReSuMe [125], SPAN [135], and Chronotron [136]. These methods approach the spiking neuron problem in different ways:

- **STDP-Based Supervision**: Training that incorporates STDP rules [125, 137].
- **Empirical Weight Computation**: Directly computing network weights [138].
- **Spike Behavior Approximation**: Approximating spike behavior to generate conventional errors [135].

A notable supervised method by Bohte et al., SpikeProp [139], introduced a way to backpropagate the gradient in SNNs. This has inspired various methods [96, 140–143]. The main challenge is that spikes are non-differentiable. Different strategies have been proposed to overcome this:

- **Spike Timing Approximation**: Approximating spike timings [140].
- **Surrogate Functions**: Replacing spikes with surrogate functions for differentiation [143].

- **Temporal Coding Exploitation**: Using time coding aspects to calculate errors [96].
- **Designing Differentiable Models**: Creating models that allow gradient evaluation [142].
- **Spatio-Temporal Framework**: Backpropagating gradients along both network depth and time dimensions [141].

Another promising method which is particularly relevant to this thesis is Spike Layer Error Reassignment (SLAYER) [24], which approximates the derivative of the spike function using a temporal credit assignment policy for backpropagating errors.

In Section 2.1.4, the concept of GD was introduced as being at the core of (most of) the conventional DL training algorithms. While such concepts work seamlessly in conventional DL, applying them to NM computing is not as straightforward. The whole algorithm is based on the calculation of the gradient of the cost function with respect to each of the NN components. In DL activation functions, which are normally derivable or quasi-derivable, are used. In SNNs, activation functions are replaced by spiking neurons which are non-derivable by nature due to their spiking function effectively resulting in Dirac's deltas. Nevertheless, the algorithms presented in this section are in fact based on GD and backpropagation. This is because, to overcome the issue, researchers were required to resort to other techniques in order to allow the calculation of the gradient and finally devised the concept of Surrogate Gradient. The Surrogate gradient acts as a proxy for spiking neurons during the backward pass, providing a surrogate derivable function or some approximations of the spiking functions around the spiking time. In this way, gradient descent and backpropagation are possible in the NM domain too.

2.2.4 Neuromorphic Hardware

Research in the NM field has been strongly led by the hardware developments. As a matter of fact, the advantages that the NM approach offers are entirely due to the possibilities that arise from te use of NM hardware. When referring to hardware, the focus is mainly on processors, which can host the computational models to process data like SNNs, and sensors, which provide an interface to perceive the world. This thesis, however, approaches the NM domain from an algorithmic perspective, thus focusing on the study of the algorithmic components

of SNNs that could be finally deployed onto NM hardware. At the same time, it aims to provide potential new directions for hardware developments by showing the applicability of SNNs to a broader range of scenarios and use cases. Nevertheless, it is important to ensure that the developed algorithms would still be able to harness the energy, latency and scaling characteristics of the underlying NM hardware. The following sections provide a brief overview of the existing NM hardware with an accent on the most relevant components.

2.2.4.1 Neuromorphic Processors

A major breakthrough in neuromorphic processing occurred in 2008 with the launch of DARPA's Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) program. This initiative led to the creation of the IBM TrueNorth chip in 2014 [144], a neuromorphic processor designed to handle large-scale neural networks while maintaining exceptionally low power consumption. TrueNorth employs a crossbar array architecture, using time-multiplexed neuron updates and limited precision for synaptic weights. Despite these design constraints, it supports one million neurons and 256 million synapses distributed across 4096 neurosynaptic cores. The development of TrueNorth signified a pivotal moment, as it marked the involvement of a major commercial entity in the neuromorphic computing domain.

Before TrueNorth, neuromorphic research had primarily been driven by academic projects, such as the Spiking Neural Network Architecture (SpiNNaker) [145]. Funded by the European Union's Human Brain Project (HBP) [146], SpiNNaker was initially developed to serve neuroscience, robotics, and computer science research. Its architecture integrates general-purpose ARM cores with tightly coupled memory on a single chip, offering high reconfigurability by running neuron models as software. The current SpiNNaker system connects over one million ARM cores, enabling the simulation of more than a billion spiking neurons with realistic synaptic connectivity (1,000 to 10,000 synapses per neuron) at 1 ms time steps. This configuration is estimated to simulate 1% of the human brain's functionality [145], with plans for SpiNNaker 2 aiming to scale up to the full brain using a 10-million-core design [147].

Another neuromorphic solution emerging from the HBP is the BrainScaleS platform [148]. This waferscale neuromorphic system combines analog and digital technologies, supporting up to 40 million synapses for 180,000 neurons. BrainScaleS leverages HiCANN neurocores interconnected on wafers, enabling precise modeling of biological neural networks at speeds exceeding real time.

Braindrop [149] represents another advancement in neuromorphic systems, developed as a successor to the NeuroGrid platform [150]. This mixed-analog-digital architecture is underpinned by the Neural Engineering Framework (NEF) [151], which simplifies the design of non-linear dynamic systems. By abstracting hardware implementation, the NEF framework mitigates challenges associated with the variability of analog neurons and eliminates the need for users to possess specialized hardware expertise.

More recently, Intel has entered the neuromorphic space with its Loihi processor [152]. Loihi is a digital neuromorphic chip that supports the evaluation of large-scale spiking neural networks (SNNs) and provides flexible neuron configurations. Designed to cater to both research and practical applications, Loihi integrates on-chip learning capabilities with support for diverse learning rules, neuron models, and coding schemes. The chip includes 128 neuromorphic cores, which collectively house 130,000 leaky integrate-and-fire neurons and 130 million synapses. Its modular architecture allows multiple chips to be combined for scalability [153]. The more recent Loihi 2 processor [101] takes this even further and introduces the possibility of programmable spiking neurons.

The DYNAP chip series [154, 155], including DYNAP-SEL, DYNAP-SE2, and DYNAP-CNN [156], further expands the neuromorphic processor landscape. The SEL and SE2 chips both feature 1,000 analog spiking neurons based on the adaptive exponential integrate-and-fire model. The SE2 includes 65,000 synapses with adjustable delays, weights, and short-term plasticity, while the SEL offers up to 80,000 reconfigurable synapses, including 8,000 with integrated spike-based learning capabilities. The DYNAP-CNN chip, designed for spiking convolutional neural networks, includes one million spiking ReLU neurons per chip and integrates seamlessly with dynamic vision sensors [156].

Collectively, these developments in the processing units have been a critical factor in the increase in interest and push in the NM field. NM chips define what can be attained in terms of algorithmic implementations on potential real-world applications, and effectively enable the low power, low latency computations that NM is known for.

2.2.4.2 Neuromorphic Sensors

Neuromorphic sensors have garnered significant interest from both academic circles and industry. This innovative sensing approach seeks to address the limitations of traditional sensors, which often generate vast amounts of redundant data, leading to excessive power consumption [157]. Examples of neuromorphic sensors include silicon retinas (event-based cameras or neuromorphic vision sensors) [158–161], silicon cochleae (neuromorphic audio sensors) [162, 163], electronic nose systems (neuromorphic olfactory sensors) [164], and robotic skin (neuromorphic tactile sensors) [165].

Among these, the most prominent is the neuromorphic vision sensor (NVS). The NVS, along with most other neuromorphic sensors, utilizes the Address Event Representation (AER) [166], a standardized method for processing sensor outputs. These bio-inspired vision devices emulate the functionality of biological retinas and operate differently from conventional cameras. Instead of recording all information at regular intervals, they produce outputs only when a change is detected.

This operation enables the sensors to capture luminosity changes at specific moments, producing a continuous temporal derivative of luminosity. Each change generates an event, $e = [x, y, ts, p]$, where x and y represent spatial coordinates, ts is the timestamp of the detected change, and $p \in \{1, -1\}$ indicates a positive or negative change in brightness. This mechanism increases signal sparsity, allows asynchronous output, and provides microsecond temporal resolution, significantly reducing power consumption and bandwidth requirements.

The NVS offers a dramatic improvement in temporal resolution, with output rates ranging from 1 to 3 orders of magnitude higher than traditional sensors (33 ms for conventional sensors versus 15 μs for event-based sensors) [160]. This capability enables the NVS to function as an ultra-high-speed camera, effectively achieving the equivalent of 66,000 frames per second for up to 800 pixels, without the need to process unchanged pixels. Another notable feature of the NVS is its high dynamic range, exceeding 140 dB compared to the 60 dB typical of conventional cameras [160, 161]. This allows event-based cameras to perform well under varying lighting conditions, including rapidly changing brightness and low-light environments where traditional cameras fail.

Neuromorphic sensors represent the best way to identify suitable application for an immediate implementation, as they provide a means to sense information in a NM way (high dynamics, sparse and event-based sensing), thus allowing the creation of an end-to-end NM system in principle. In this thesis, NVS have

been indirectly used through two NM sensor-based datasets in Chapter 4. Other chapters explore other avenues of applications and information encoding, but can be viewed as a starting point to develop future NM sensors so as to enable such applications to be NM from end to end.

2.2.5 Frameworks for SNNs

The development of Deep Learning algorithms was greatly eased by the introduction of purposely developed software packages. These packages, or frameworks, usually offer a wide range of software tools that aim to speed up the development of ML pipelines as well as make the algorithms available to a larger audience. When referring to conventional DL, i.e. non Neuromorphic, several famous libraries exist, such as TensorFlow (TF) [167], PyTorch [168] or Caffe [169]. The field of Neuromorphic engineering has recently seen the emergence of several new software frameworks thanks to the renewed interest in its potential. However, these frameworks are often in an early development stage when compared to their conventional DL counterpart, being limited in the tools they offer, their documentation, and the support from the community. Some more established frameworks also exist, but they are often directed towards particular communities and use cases [15], or they are neuroscience-oriented frameworks rather than NM-ML development tools. Furthermore, effective data science algorithms that can close the gap with other conventional methodologies still need to be developed. Indeed, research highlights that, thanks to their use of sparse event-based communications, algorithms employing SNNs can be shown to be more energy efficient than others based on conventional CNNs [170], however, they are not as effective on ML tasks in terms of accuracy. Hence the importance of having good software frameworks that enable customization, simulation and deployment of SNNs. This requires combining a number of key elements into a pipeline such as learning rules, connectivity patterns, and spiking neurons. Regarding the spiking neurons, emerging NM chips such as Loihi 2 [101] allow the use of customized models. It has been shown in the literature that different types of neuron models can solve certain tasks more effectively than other models [17, 76]. Therefore it can be beneficial for researchers to use a framework that enables seamless experimentation with different types of neurons.

In the literature, when presenting a new software framework, authors often provide a brief report on other similar works, and draw comparisons with them [20, 171]. In these instances, differences in terms of offered features are highlighted, as well as the advantages of using the newly presented software over the

existing ones. Other works specifically focus on reviewing the existing frameworks for the development of SNNs. One example is given by [172], where the authors make a subdivision of the software packages into three main groups depending on whether they are NM chips toolchains, SNN simulation frameworks or frameworks that integrate SNNs and DNNs. Another work [170] gives an introductory overview of SNNs and then reviews some prominent simulation frameworks. The authors also define a simple classification task and compare the accuracy and execution time obtained by using the different frameworks. These previous two works consider frameworks regardless of their research orientation, i.e. they consider both neuroscience-oriented and data science-oriented frameworks. In the following sections, nine of the most prominent data science-oriented software frameworks for the development of SNNs will be briefly reviewed, with a particular highlight on available spiking neuron models and learning rules. The review focuses on data science-oriented frameworks because they are more relevant to the focus of this manuscript, and because the aim is to provide a more specific reference to frameworks that are useful to develop NM applications in this domain.

2.2.5.1 Software Frameworks

Many of the software libraries for the development of SNNs are oriented toward the needs of the neuroscience and neurobiology fields [170]. Because SNNs process inputs and communicate information in a way similar to the human brain, they are particularly well suited for simulations of brain areas activations. Nevertheless, the recent emergence of NM engineering as a field for developing ML algorithms has highlighted the need for suitable frameworks. Researchers have been active in the implementation of software tools that can ease the development of NM algorithms in this direction. In the following sections, nine amongst them are briefly reviewed, and a summary of some of their prominent features is reported in Table 2.2.

Tab. 2.2. Key elements of the reviewed frameworks. The "A-" stands for adaptive, whereas "H-" stand for heterogeneous.

Framework	Nengo	Lava	SNN Toolbox	Norse	PySNN	snnTorch	SpikingJelly	BindsNet	SpykeTorch
Spiking Neurons	LIF A-LIF IZ	LIF RF* A-LIF* A-RF* A-IZ* $\Sigma - \Delta$	IF	LIF AdEx EIF IZ LSNN	IF LIF A-LIF	LIF Recurrent LIF 2nd Order LIF LSNN	IF LIF pLIF QIF EIF	IF LIF A-LIF IZ SRM	IF LIF** QIF** EIF** AdEx** IZ** H-Neurons**
Learning Rules	Oja BCM BP	SLAYER STDP 3-Factor	Pre-trained	SuperSpike STDP	STDP MSTDP MSTDPET	BPTT RTRL	BP	STDP Hebbian MSTDPET	STDP R-STDP
Conversion from	TF/Keras	PyTorch	TF/Keras PyTorch Caffe Lasagne	-	-	-	PyTorch	PyTorch	-
Destination Backend/Platform	Loihi FPGA SpiNNaker MPI CPU/GPU	Loihi CPU/GPU	SpiNNaker Loihi pyNN Brian2 MegaSim	CPU/GPU	CPU/GPU	CPU/GPU	CPU/GPU	CPU/GPU	CPU/GPU

* Only available in Lava-DL.
** Added in this work (see Appendix A).

2.2.5.1.1 Nengo Nengo [173] is a Python package for building and deploying neural networks. It is composed of several sub-packages to be used in case of different needs and destination platforms. NengoDL is to be used when aiming to convert a CNN built using TF/Keras into its Nengo spiking version. NengoLoihi allows to deploy NNs natively built in the Nengo Core package onto Loihi chips. Other packages are NengoFPGA, NengoSpiNNaker, NengoOCL and NengoMPI. Nengo builds on top of a theoretical framework called the Neural Engineering Framework (NEF) [174]. Computations are based on the three principles of the NEF: neural representation, transformation, and neural dynamics. Neurons in Nengo are organized in Ensembles, and different types of neuron models are available, among which the LIF[70], and IZ[16] models. Connections between ensembles are designed to allow a transformation of the information from one ensemble to another. Training in Nengo is possible with the Oja [175], BCM [176] and backpropagation learning rules. Using Nengo as a tool for the development of SNNs has the main advantage of having the possibility to target a wide variety of backends and to convert conventional DNNs into a spiking equivalent [170]. Nengo also allows for a certain degree of customization of the components; however, it remains very oriented towards the NEF structure.

2.2.5.1.2 SNN Toolbox SNN Toolbox [62] provides a set of tools to perform automated conversion from conventional ANN models into SNNs. Conversion is possible from three different DL frameworks, namely TF/Keras, PyTorch, Caffe and Lasagne [177]. The framework supports conversion to models for PyNN [178], Brian2 [179], MegaSim [180], SpiNNaker [181], and Loihi [182] where the SNN can be simulated or deployed. However, depending on the components used in the original ANN, some of the target platforms might not be available. During the conversion phase, IF neurons are used for a one-to-one substitution. These are then tuned so that their mean firing rate approximates the activation of the corresponding neuron in the original ANN. Neural networks must be pre-trained in their original framework. Tuning conversion parameters and performing inference is possible either through the command line or through a simple GUI.

2.2.5.1.3 Lava Lava [183] is a relatively recent framework built by Intel's Neuromorphic Computing Lab (NCL). The framework results from an evolution from the Nx SDK software for Loihi chips, but aims to target other hardware platforms as well. Lava is composed of 4 main packages, namely Lava (core), Lava-DL, Lava Dynamic Neural Fields (DNF) and Lava Optimization. The current state of the platform includes the development of deep SNNs trained with SLAYER [24], and of SNNs converted from PyTorch. On-chip training through SLAYER is currently not available. Instead, models need to be trained off-chip, and weights must be exported to be used within the Lava core package. Within Lava-DL, a number of neuron models are defined, such as the LIF, RF [75], RF Izhikevich, Adaptive LIF [12], Adaptive RF, and Sigma-Delta [184] modulation models. The core package currently supports LIF and Sigma-Delta modulation neurons. Recent developments in the framework have seen the implementation of on-chip learning functionalities through STDP and customized 3-factor learning rules.

2.2.5.1.4 PyTorch-based Frameworks

Norse

Norse [185] is a relatively recent PyTorch-based framework. It was developed with the aim of easing the construction of SNNs for ML solutions. This framework offers a wide range of neuron models, such as the LIF, LIF variants and extensions, and Izhikevich's model. It also provides a LSNN [186], a spiking version of the LSTM (Long Short-Term Memory) [187]. Norse has a functional programming style. Neurons are mainly implemented as functions and do not hold an internal state. Instead, the previous state of the neuron needs to be provided as an argument at each iteration. The framework mainly allows for two types of learning: STDP [134], and SuperSpike [143]. Therefore, both local unsupervised learning and surrogate gradient learning are possible. Overall, Norse provides a good degree of flexibility and allows leveraging all of the features of PyTorch, such as GPU acceleration.

PySNN

PySNN [188] is another framework based on PyTorch aimed at developing ML algorithms. Similarly to Nengo, connections between two neurons are modelled as separate objects that have properties and can affect the transmission of a signal. For instance, they can explicitly account for connection delays. Neuron models in PySNN embed the concept of spike trace, which can be used for learning purposes. Some available neuron models are the IF, LIF and ALIF. Concerning the learning rules, it is possible to use either STDP or MSTDPET (Modulated STDP with Eligibility Traces) [189]. The framework also provides some useful utilities to load some NM datasets. A downside of using PySNN is that the documentation is not complete.

SnnTorch

SnnTorch [190] also bases its architecture on PyTorch. Connectivity between layers is enabled by leveraging PyTorch standard layers. Spiking neurons are thought to be used as intermediate layers between these. Spiking neurons are modelled as classes that hold their own internal state. Available models include LIF-based models, second-order LIF models, recurrent LIF models, and LSTM memory cells. Learning in snnTorch takes place with BP Through Time (BPTT) using surrogate gradient functions to calculate the gradient of the spiking neurons. The framework also offers the possibility to use a Real-Time Recurrent Learning (RTRL) rule, which applies weight updates at each time step, rather than at

the end of a sequence of inputs. The network output can be interpreted using both a rate-based approach and a time-to-first-spike (TTFS) approach. Finally, snnTorch provides access to the N-MNIST [191], DVS Gestures [192], and the Spiking Heidelberg Digits [193] datasets, and includes useful network activity visualization tools.

SpikingJelly

SpikingJelly [194] is a framework using PyTorch as a backend and adopting its coding style throughout. It provides implementations of IF, LIF, parametric LIF (pLIF), QIF, and Exponential IF neuron [77] models. The firing of neurons in SpikingJelly is approximated by a surrogate function (such as the sigmoid) that allows differentiation. The framework provides several utilities to read NM and non-NM datasets. Concerning the NM datasets, it is possible to both read them with a fixed integration time-window and with a fixed number of frames. Among the available datasets, there are the CIFAR10-DVS [195] dataset, the DVS Gestures dataset, the N-Caltech101 [191] dataset, and the N-MNIST dataset. Finally, SpikingJelly also provides functionality for ANN to SNN conversion from PyTorch.

BindsNet

BindsNet [171] is a library for the development of biologically inspired SNNs. Despite having PyTorch as a backend, the coding style differs slightly. Execution is implemented by running the network for a certain amount of time on some input rather than explicitly looping through the dataset. BindsNet supports several types of neuron models: IF, LIF, LIF with adaptive thresholds, Izhikevich's, and Spike Response Model (SRM)-based [12] models. Connections are modelled explicitly and link one node of the network with another. Recurrent connections are also possible. The provided learning rules are biologically inspired and can be either two-factor (STDP or Hebbian) or three-factor (MSTDPET); hence no BP-based learning rule is proposed. Through sub-classing, it is possible to customize neurons, input encoding and learning rules. The framework also provides utility tools to load datasets, such as the spoken MNIST, and DAVIS [160] camera-based datasets. Finally, BindsNet includes a conversion system to convert neural networks developed in PyTorch into SNNs.

SpykeTorch

SpykeTorch [20] is PyTorch-based library for building SNNs with at most one spike per neuron. This means that for each sequence of inputs, each neuron is allowed to fire only once. Because of this, tensor operations can be easily used to compute neuron activations. Because NM data includes the concept of time, what is normally treated as the batch dimension in PyTorch is interpreted as the time dimension in SpykeTorch. The framework is built to support STDP and Reward-modulated STDP (R-STDP) with a k-Winner Takes All (kWTA) paradigm to perform training only on the top k neurons at a time, and using convolutions as a connection scheme. The only available neuron model is the IF, which is provided as a function. Finally, the framework provides functionalities to encode non-NM input through difference of Gaussians and intensity to latency transforms, as well as some inhibition functions.

2.3 Conclusions

In this Chapter, fundamental elements that compose the field of conventional Neural Networks and NM computing have been laid out. The initial excursus on conventional Neural Networks provides the ground to understanding the differences with their spiking counterparts, and to understanding some key components utilized in later chapters. Topics ranging from types of layers and activation functions to the backpropagation algorithm are covered, helping to understand the foundations on which Spiking Neural Networks have been developed. Neural network structures based on both fully connected and convolutional layers are presented. The concept of activation functions is closely related to the role of spiking neurons in an SNN. Loss functions, which are a key contribution presented in Chapter 5, are also discussed. Concerning NM computing, Spiking Neuron models within SNNs and SCNNs represent a core component that effectively differentiates them from conventional ANNs and CNNs. A review of the most relevant ones is presented, as well as a review of works focusing on studying their properties. This is particularly relevant for Chapter 4, as such works are critically reviewed by highlighting their focus as being primarily on the neurobiological aspects of spiking neurons. Due to the increased interest in applying NM concepts to engineering problems, this emphasizes the need for further understanding of spiking neurons as a way to enhance the effectiveness of said approaches. Moving on, neural coding algorithms are an ever-present concept in designing SNN pipelines. These are particularly relevant for Chapters 5 and 6,

and the key concepts behind them are thus presented. Learning rules are what enable an NN to learn from some given data. In NM computing they take both inspiration from biology and from teachings from conventional DL. Throughout the following Chapters, both types will be employed, therefore relevant examples are reported. Finally, simulations are the starting point when designing an NM pipeline. Such simulations are enabled by software frameworks, which help speed up the overall process. Because of this, a review of the most prominent ones is also reported in the sections above.

Chapter 3

Neuromorphic Applications in the Time Series and EMG Domains

3.1 Introduction

When delving into the diverse application domains of NM computing, researchers predominantly focus on tasks such as image classification [196], object detection [197] and tracking [198], SLAM [199], and optical flow [10]. These tasks represent the cornerstone of current research efforts in the field. However, the scope of NM computing is continually expanding as researchers seek to uncover novel applications that can harness the unique advantages offered by NM paradigms.

This chapter provides a brief introduction to two promising applications that have been explored in this thesis as potential research avenues for NM-based solutions. First, the chapter discusses time series forecasting, presenting the most conventional methods for processing data and performing forecasting. The chapter also reviews the relevant literature that underpins the work detailed in Chapter 5, offering a comprehensive background to contextualize the research findings. Subsequently, the chapter introduces the Electromyography (EMG) gesture classification task. This section highlights the use of the Ninapro dataset [25], which is a pivotal resource utilized in Chapter 6. Additionally, the chapter references significant literature related to solving the EMG gesture classification task, thereby providing a thorough foundation for understanding the approaches and methodologies applied in this area of research.

In summary, this chapter aims to present a broad overview of these two emerging applications, underscoring their potential to benefit from NM computing

paradigms and setting the stage for the detailed discussions and analyses presented in the subsequent chapters.

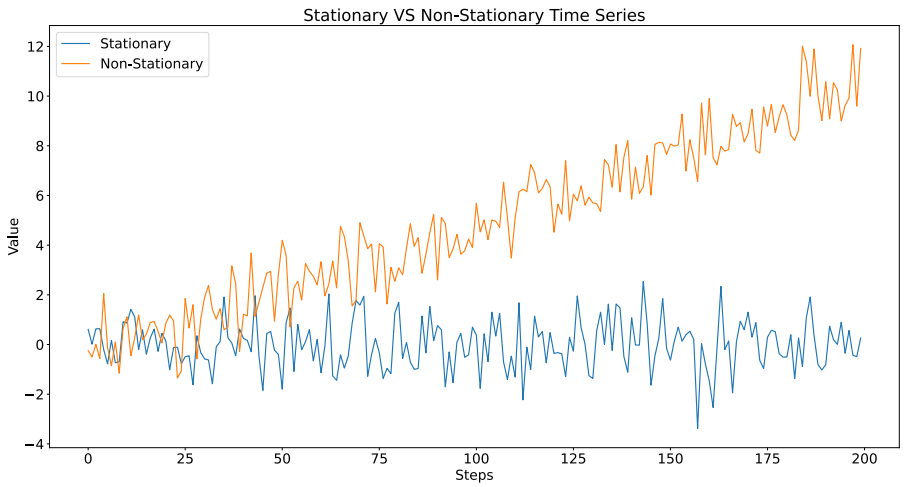


Fig. 3.1: Example of a Stationary and a Non-Stationary signal). The stationary (blue line) signal is white noise, while the non-stationary (orange line) signal exhibits a linear trend.

3.2 Time Series Forecasting

Time series forecasting is essential for predicting future values based on historical data, and it has applications across various domains such as finance, economics, meteorology, and engineering [200–202]. Time series are composed of data points indexed by time, and can be broadly classified into stationary and non-stationary categories [203]. An example of this can be found in Fig. 3.1. Stationary time series are characterized by statistical properties such as mean, variance, and autocorrelation that remain constant over time. This consistency simplifies the analysis and forecasting processes, making stationary time series ideal for applying many statistical methods. Non-stationary time series, on the other hand, exhibit changing statistical properties over time. These changes can manifest as trends (see Fig. 3.1 for an example of a linear trend), seasonal patterns, or structural breaks, which complicate the modelling process . Non-stationary time series can lead to unreliable and misleading results if not appropriately addressed. Therefore, transforming non-stationary time series into stationary ones is a crucial step in time series analysis. Techniques such as differencing, detrending, and seasonal adjustment are commonly used to achieve stationarity [203]. The methods employed for time series forecasting have experienced a profound evolution

over the years, reflecting a significant shift from traditional statistical techniques to more advanced modern methodologies [201]. Initially, forecasting relied heavily on classical statistical approaches, which were grounded in fundamental mathematical principles and aimed at identifying patterns and trends within historical data [200, 203]. However, as the field progressed, the focus began to shift towards incorporating modern machine learning techniques [202], which introduced new algorithms and computational methods capable of handling more complex and dynamic data sets. The latest advancements in this domain have further advanced the state of forecasting by integrating sophisticated neural network models, which leverage deep learning architectures to capture intricate patterns and dependencies within the data [202].

3.2.1 Classical Methods

Classical methods of time series forecasting are well-established and widely used due to their simplicity and interpretability. These methods include the Autoregressive Integrated Moving Average (ARIMA) [200], which combines autoregression, differencing, and moving averages to model time series data. It is possible to define an ARIMA(p, d, q) model as the following (adapted from [203]):

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t, \tag{3.1}$$

where y'_t is the differenced series, ϕ_p are the autoregressive model parameters, θ_q are the moving average model parameters, ϵ_t is white noise, and p, d, and q denote the order of the autoregressive part, of the differencing (of y) and of the moving average part respectively. ARIMA is highly interpretable and effective for univariate time series that are stationary or can be made stationary through differencing. It is particularly useful for capturing linear patterns and trends in data. However, ARIMA requires the time series to be stationary, necessitating pre-processing steps like differencing, and may struggle with capturing non-linear relationships and complex patterns in the data [204]. The Seasonal ARIMA (SARIMA) is a version of the ARIMA that is able to account for and model seasonal effects in the data.

Another classical method is Exponential Smoothing with error terms (ETS - Error, Trend, Seasonal), known for its flexibility in handling a wide range of time series patterns, including trends and seasonality. Many different types of ETS models are possible depending on the choice of the ETS components. A simple one is the exponential smoothing with additive errors ETS(A,N,N) (A =

Additive, N = None) [203]:

$$\begin{aligned} \text{Forecast equation} \quad & \hat{y}_{t+1|t} = \ell_t \\ \text{Smoothing equation} \quad & \ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}, \end{aligned} \tag{3.2}$$

where $\hat{y}_{t+1|t}$ represents the forecast value at time t+1, ℓ_t is the smoothed value at time, and α is a smoothing parameter considering the previous value of time series y_t and the previously smoothed value. ETS models are particularly effective for short-term forecasting and are easy to implement. However, they may not perform well with long-term forecasts or in the presence of complex, non-linear relationships, and require careful selection of smoothing parameters [205]. The Holt-Winters method extends exponential smoothing to capture both trend and seasonality [206]. It is straightforward to implement and effective for data with strong seasonal patterns. However, it may not perform well for non-linear trends or when the seasonal pattern changes over time.

The Seasonal Decomposition using Loess (STL) [207] of Time Series is a robust method for decomposing time series data into seasonal, trend, and residual components. This non-parametric approach offers flexibility and adaptability to various types of data. However, STL can be computationally intensive, particularly for large datasets, and requires a substantial amount of historical data to accurately capture the seasonal component. Despite these challenges, STL remains a versatile and robust time series decomposition method [208].

Finally, Wavelet-based and Fourier forecasting methods provide a useful way to enhance forecasting accuracy by offering complementary perspectives on time series data [209, 210]. Wavelet methods excel at handling non-stationary data and capturing localized features across multiple scales, making them adept at detecting transient patterns and anomalies. In contrast, Fourier methods are powerful for identifying and modeling periodic behaviors in stationary data through frequency decomposition.

3.2.2 Conventional Machine Learning

The advancement of machine learning has significantly impacted time series forecasting by enabling the analysis of large datasets and intricate patterns that classical methods struggle to capture. Recurrent Neural Networks (RNNs) are particularly notable for their ability to handle sequential data and effectively capture temporal dependencies [211]. RNNs maintain a hidden state that retains information from previous inputs, making them adept at modelling time series

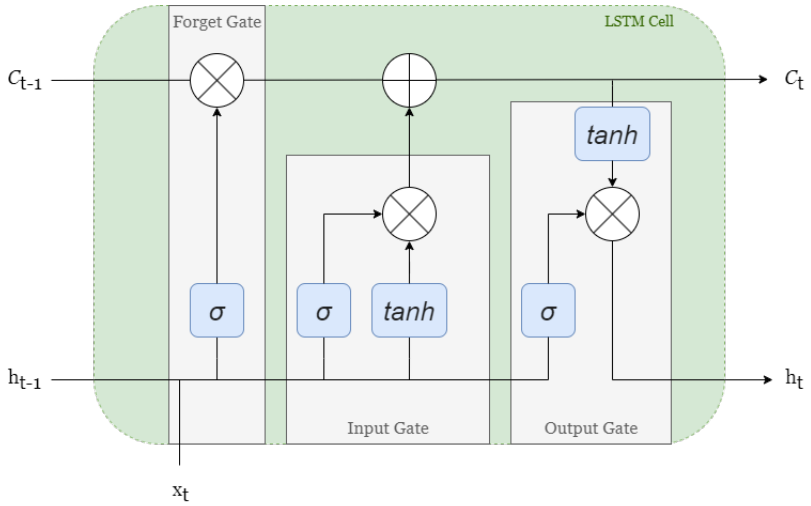


Fig. 3.2: Schematics of a typical LSTM cell, adapted from [213].

with complex temporal patterns. However, challenges such as vanishing gradients can hinder their performance in capturing long-term dependencies [212]. Moreover, the training of RNNs requires substantial computational resources, which can be a limiting factor in practical applications. Long Short-Term Memory Networks (LSTMs) address the limitations of traditional RNNs by incorporating memory cells that store information for longer periods [213, 214]. These LSTM cells are internally composed of three "gates": the forget, the input, and the output gates. A schematic of this is reported in Fig. 3.2, which shows the use of the cell's state (C_t) for retention of long-term information, and of the hidden state (h_t) for short-term information propagation. To be more specific, these are internally calculated as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{3.3}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{3.4}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{3.5}$$

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C), \tag{3.6}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{3.7}$$

$$h_t = \tanh(C_t) * o_t, \tag{3.8}$$

where i_t, f_t, o_t, and $\tilde{C}_t$ are the input gate, forget gate, output gate and activation vectors respectively, $W_{<g>}$ refers to the set of weights associated with each variable, $b_{<g>}$ to the bias for each variable, x_t is the input at time t, h_t and C_t the hidden state and Cell state at time t. Each gate is thus equipped with a set of learnable weights, that enable the LSTM to learn what to forget, what to

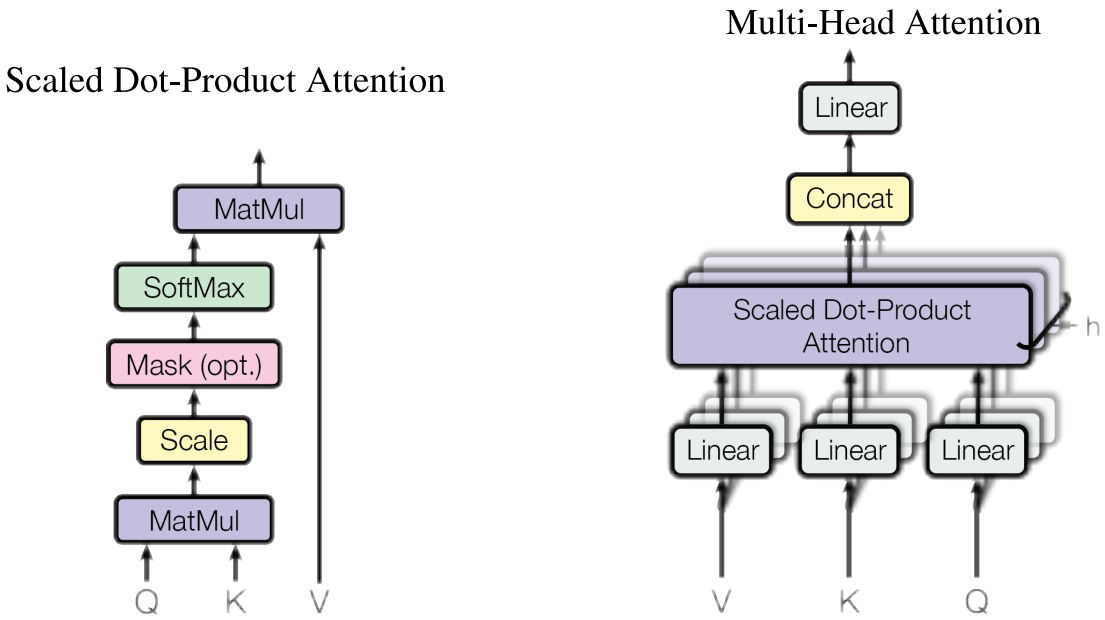


Fig. 3.3: Schematics of the dot-product attention (left), and the multi-head attention module (right), adopted from [221].

retain, and what to propagate forward [215]. This makes them highly effective for capturing long-term dependencies in time series data. Despite their advantages, LSTMs are computationally intensive [216–218], require a large amount of training data, and can be difficult to interpret compared to classical models.

Convolutional Neural Networks (CNNs), originally developed for image processing, have been adapted for time series forecasting by treating the data as a spatial sequence [219]. CNNs can automatically learn hierarchical features and are efficient in capturing local patterns. However, they may not be as effective for capturing long-range dependencies compared to RNNs or LSTMs and also require significant computational resources.

Transformers, known for their success in natural language processing and computer vision tasks, are now being increasingly explored for time series forecasting applications. These models leverage self-attention mechanisms to capture long-range dependencies and intricate temporal patterns [220]. Differently from LSTMs and RNNs, they do not have any recurrent unit, and process data sequences at once. Data points in a sequence are first tokenized, i.e. transformed into a numerical representation (e.g., from a text input), and then contextualized by means of a multi-head attention mechanism (see Fig. 3.3) [221].

This is roughly based on the calculation of the following dot product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{3.9}$$

where Q, K, and V are the query, keys and values vectors respectively obtained via the application of a fully connected layer to the input sequence, d_k is the dimensionality of keys, queries and values, and softmax denotes the following operator:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_i^N e^{x_i}}. \tag{3.10}$$

More in general, so-called Large Language Models (LLMs) in which attention-based mechanisms are deeply rooted, are a transformative technology that redefine human-computer interaction and drive innovation across industries [222]. They are groundbreaking tools that democratize knowledge and can provide efficient solutions to several types of problems in seconds. However, such transformative power comes at a great cost, both in economic and environmental terms. Several studies have in fact shown that the carbon footprint of LLMs is a matter of concern [223–225]. LLMs have in fact grown to having trillions of learnable parameters and setting up one entails weeks of GPU-intensive training, storage costs (one LLM can occupy several gygabytes), and inference costs once its been deployed. One example is provided by BLOOM [224], a 176-billion parameters model which is estimated to have emitted approximately 50.5 tonnes of CO_2 during its final training phase alone. The specific task of an LLM is to process a sequence of tokens (words) in order to perform some actions. This is in essence a special case of time series forecasting, which makes them relevant to the topics treated in this thesis. As a matter of fact, some time series forecasting is also performed as part of the training of some LLMs [222]. They are normally faster to train than LSTMs and RNNs, however, their complexity and computational demands of pose challenges, requiring substantial amounts of training data to achieve optimal performance and a vast amount of hardware resources that can often only be provided by server farms, with extensive economical and environmental costs.

In addition to deep learning techniques, classical machine learning methods like Support Vector Machines (SVMs) are also used in time series forecasting. SVMs are classical machine learning methods that have been widely used in time series forecasting due to their effectiveness in classification and regression tasks by finding optimal hyperplanes to separate classes or predict continuous values [226, 227]. However, SVMs can be challenging to scale with large datasets due to the fact that the training kernel matrix grows quadratically with the size of the data set [228] and often require careful tuning of kernel functions and hyperparameters to achieve optimal performance

3.2.3 Neuromorphic Computing

Within the context of Neuromorphic Computing and Spiking Neural Networks, time series forecasting has not garnered as much attention as other types of tasks; nevertheless, there is an interest in applying NM concepts to this domain. One notable work [229] delves into the application of a specific type of SNN, the Polychronous Spiking Network, for financial time series prediction. This demonstrates the feasibility and effectiveness of SNNs in handling time series data, particularly in the domain of financial forecasting. The authors in [230] utilize an STDP-trained SNN to predict price spikes in price time series and deal with high-frequency data, demonstrating encouraging results. In [231], the authors compare conventional DL-based solutions with their spiking implementations on financial time series, concluding that better training systems are required to bridge the gap between the two, which sees the SNNs achieving lower-lever performance.
A crucial element to consider when attempting to apply SNNs to time series data is the encoding. Conventional time series are composed of floating-point numbers, which are not normally suited for SNN processing [232]. This is a common and required step whenever dealing with data that does not yet exist in an event-based NM format. In the works presented above, the authors adopt different methodologies to encode the data into spikes. In [230], Poisson encoding is utilized to transform real-valued data into random sequences of spikes; [229] employs a one-to-one encoding system between discretized time series values and encoding neurons, a potentially effective way that nonetheless suffers from potential scaling issues. As a matter of fact, to overcome this the authors bound the number of encoding neurons to 100 and were required to perform a min-max scaling and rounding as pre-processing steps so that the data would fit well with their encoding choice. Other interesting works focus primarily on the importance of the encoding system. In [232], the authors propose an interesting encoding scheme that accounts for multivariate time series and transforms such series into spatial-temporal spike patterns. They achieve this by means of a population of randomly initialised CuBa LIF neurons (see Section 2.2.1.2.5) that fire whenever input is received that is affine to the neurons' synaptic weights. Whilst this represents an improvement with respect to rate coding-based methods [232], their encoding layer requires to be pre-optimized on the dataset to improve data representation power and relies on the assumption that CuBa neurons can be effective interpreters of the features of the time series. In [233] the authors propose a Single-Modal Pulse Encoding Module, composed of an image feature extractor (which considers plots of time series) and an SNN LIF-based module. The over-

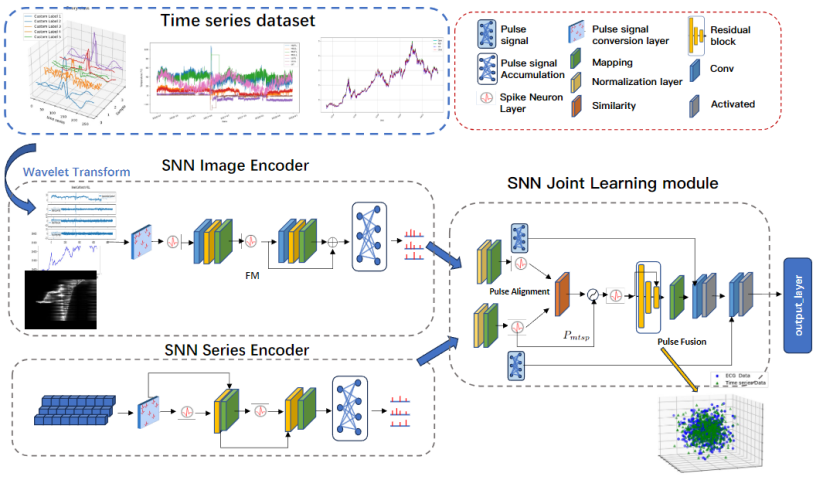


Fig. 3.4: Encoding scheme as reported in [233]. The time series undergoes preprocessing transformations such as WT, then two independent SNNs parse it as an image and as a sequence of values to produce encoding spikes. Said spikes are finally used to produce a prediction.

all approach seems to attain state-of-the-art accuracy levels, thus highlighting the potential of SNNs in the time series domain. Nevertheless, the encoding scheme could potentially introduce a non-negligible overhead, as it requires several feature extraction steps before the encoding can finally take place. This is also testified by Fig. 3.4, where the encoding schematics are reported. Here, the time series undergo transformations and is then parsed both as an image and a sequence of values by two independent encoding SNNs. An encoding system based on the interval between two spikes is proposed in [234]. This encodes the real-valued information as the size of the time interval between two consecutive spikes, and the same approach is employed to encode the output information from the network. As such, small values would result in sequences of close-by spikes in time, while large values would see more time-distanced spikes. The proposed system, completed by an evolutionary algorithm for training, shows promising results by reporting a lower error on the datasets used for training with respect to other models such as an MLP and an RNN. However, the encoding system can potentially suffer from prolonged latency periods due to the larger values needing to be encoded by larger inter-spike intervals. More broadly, the authors in [235] make a thorough review of spike encoding systems (not necessarily related to time series forecasting), and demonstrate how the use of the correct encoding algorithm can help bridge the performance gap between Spiking Neural Networks and conventional Artificial Neural Networks.

Finally, notable efforts also relate to the transposition of LLM-based solution into a NM context. In [236], the authors succesfully build and train a spike-

based version of GPT [237], one of the first LLMs developed, and demonstrate it's ability to not only match the performance of its counterpart, but also to have lower energy consumption levels by up to two orders of magnitude. An other effort was presented in [238], where the authors expand on the spike-based LLM works and introduce an in-depth analysis and alternative methodologies to improve the time complexity, energy consumption and overall efficiency of these types of models. Such solutions offer an outstanding alternative to standard LLMs, which have been previously discussed to be extremely costly. Nevertheless, they still require large amounts of computations, and are still required to perform some operations non-neuromorphically. A full end-to-end NM solution could take the advantages brought by spike-based sequence analysis even further.

3.2.4 Energy Load Datasets

The breadth of application possibilities for time series forecasting and analysis extends to a large number of different domains. As discussed in the previous section, time series forecasting has become an indispensable tool across a wide array of application domains. Its versatility stems from the ability to predict future behavior based on historical data, enabling informed decision-making and optimization in numerous fields. This thesis narrows its scope to energy load forecasting, a critical area within the energy sector that directly impacts operational efficiency, resource allocation, and sustainability efforts. Accurate energy load forecasts are pivotal for balancing supply and demand, minimizing costs, and supporting the integration of renewable energy sources into the grid, thus providing energy providers with the right tools to make choices that are beneficial to the environment.

Furthermore, the two specific datasets that have been focused on serve the purpose of the research carried out in Chapter 5 well. These two datasets are the Panama Short-term electricity load forecasting (Panama) [21] and the Electricity Transformer Temperature with one-hour resolution (ETTh1) [239] datasets. The Panama dataset was built on top of publicly available data sources, with data available from January 2015 until June 2020. The dataset presents visible periodicity in the load, as depicted in Fig. 3.5. The figure shows the electricity load week-by-week and, in particular, highlights peak usages within certain hours of the day, with daily periodicity. Nevertheless, the specific energy load within each day and corresponding days across different weeks varies. In Fig. 3.6, it can be seen how there appears to be a positive correlation between the energy load and the temperature in Celsius degrees. This not only hints to the variability within

each day, but also to the variability in demand peaks across different months of the year.

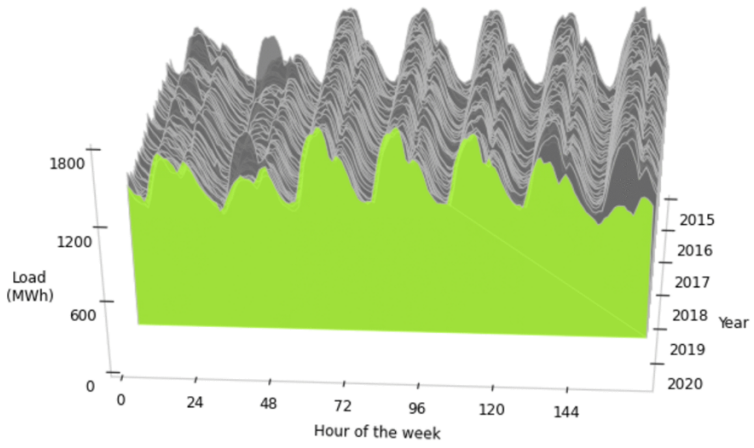


Fig. 3.5: Panama energy demand week-by-week across years 2015-2020.

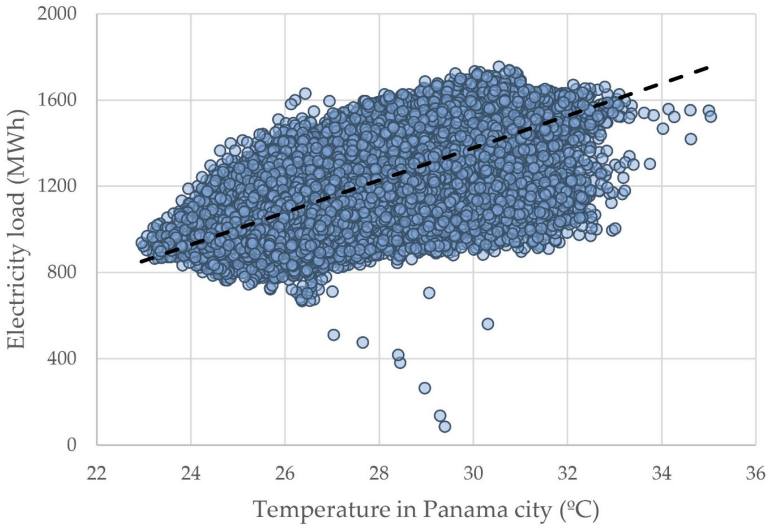


Fig. 3.6: Correlation between Temperature and Electricity Load in Panama dataset, adopted from [21]. An increase in 1 °C correlates to an increase of 74.8 MWh in energy demand.

The ETTh1 dataset, instead, focuses on the prediction of the electricity transformer's oil temperature as an indirect measure of possible peaks in the demand. That is because electricity demand forecasting is a non-trivial task, and companies often need to make decisions based on numbers that are much more inflated than the real world ones in order to be safe [23]. Forecasts based on the transformer oil temperature, on the other hand, allow to monitor the health of the transformer itself, but can also better inform decisions to balance the electricity load across different transformers and energy sources. An excerpt from the

ETTh1 can be found in Fig. 3.7. From the plot, the data have a highly variable outlook within short time periods, but also an overall decaying trend across the reported time span. Finally, the ETTh1 dataset has become a very popular option for energy-related time series forecasting, and can thus serve as a good benchmarking platform for future studies.

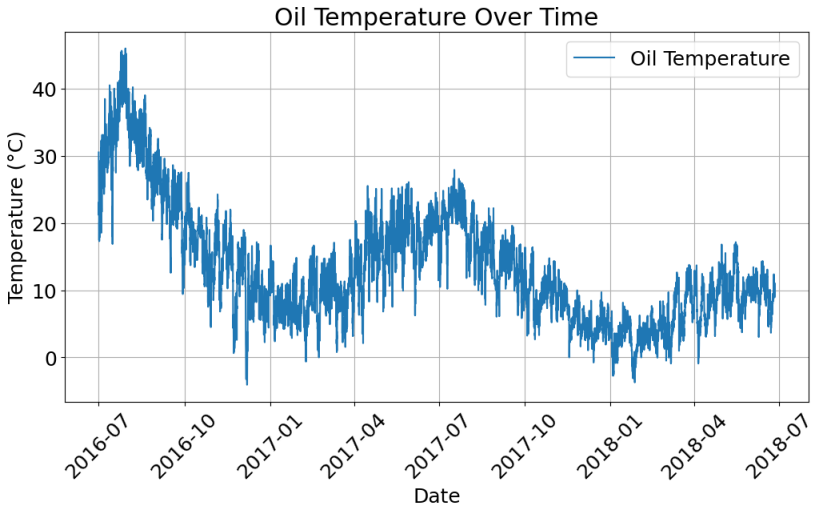


Fig. 3.7: Excerpt from the ETTh1 dataset. The plot depicts the Oil Temperature (target variable).

3.3 EMG Gesture Classification

EMG signals are commonly used in various clinical and biomedical applications, particularly in myoelectric prosthetics control for muscle movement classification [10, 240]. EMG measures the electrical impulses that muscle cells generate when they contract and relax. This electrical activity, recorded by means of a series of accurately positioned sensors (in the case of surface EMG (sEMG), this is on the skin), is a direct reflection of the underlying muscle activity and offers insights into muscle function. They are crucial for enabling precise control of prosthetic limbs by interpreting the electrical activity generated by muscle contractions. Through this application, individuals with limb loss can achieve more natural and intuitive control, greatly enhancing their quality of life. However, the processing of these signals is not trivial as they are susceptible to noise and interferences [240, 241]. Raw EMG signals are thus put through a pipeline of operations before classification can take place. The first step typically involves preprocessing and windowing [242]. Preprocessing involves the use of normalization and rectification techniques [243] among the others, as well as filtering of unwanted frequency

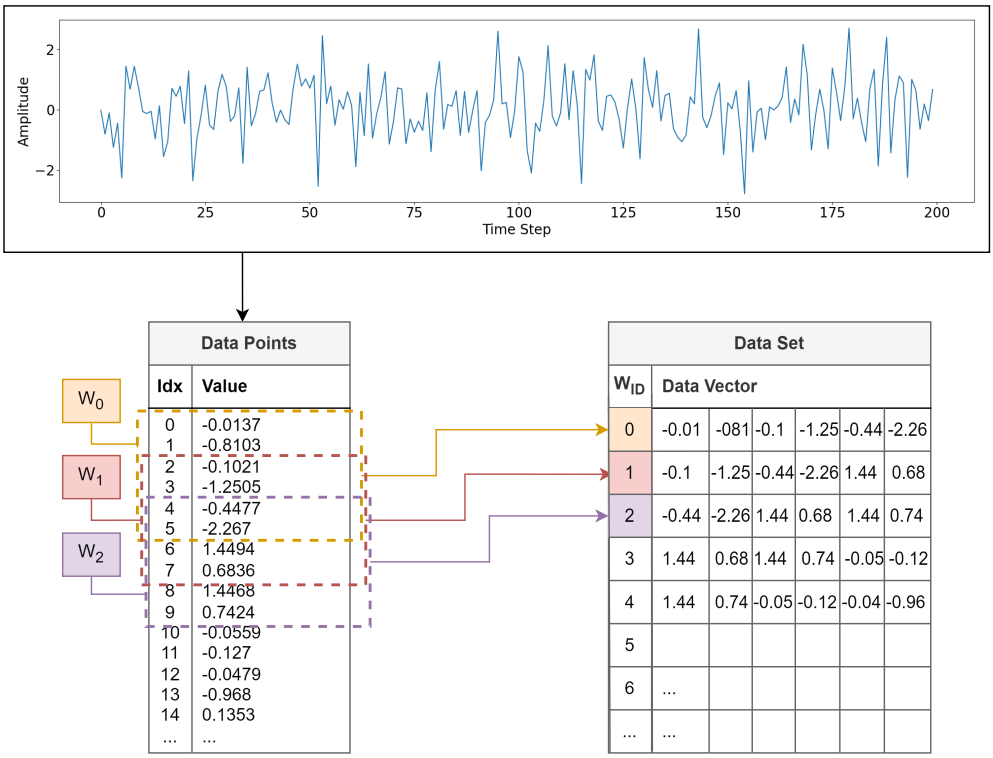


Fig. 3.8: Example of simple windowing. A window of size 5 is used to scan through the data points with an overlap of 2 time steps. Every data point appearing in the window from time to time is then added to an array of values, thus creating a sequence of vectors forming a data set.

bands (e.g. the 50 Hz component from the mains supply when present). The windowing process has been deemed of utmost importance for useful information to be visible. It consists of chunking the signal into smaller windows of a set length in time (e.g. 260 ms) before performing any feature extraction operation. Furthermore, windows are normally selected with a percentage of overlap with one another so as to be able to cover more features in the data, and to potentially obtain more frequent classification outputs in shorter periods of time [242]. A simple example of this can be found in Fig. 3.8, where a window of size 5 (time steps) and overlap 2 is used to create a data set starting from a single series of data points.

Following the initial preprocessing, a feature extraction step is usually applied. This involves using a number of techniques like Mean Absolute Value (MAV), Root Mean Square (RMS), Slope Sign Change (SSC) [243], to extract information from each window and stack them together into a single feature vector. Depending on the number of features extracted, the pipeline might require the introduction of a dimensionality reduction step too [244]. Examples of this are given by Principal Component Analysis and Linear Discriminant Analysis,

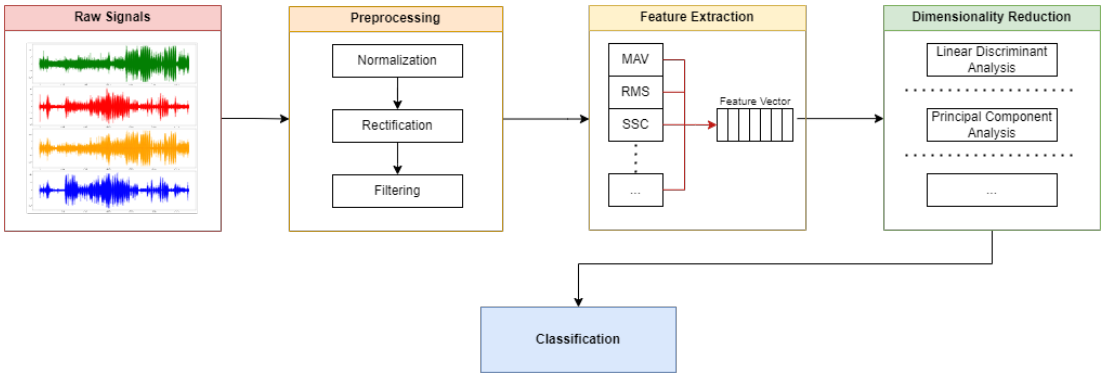
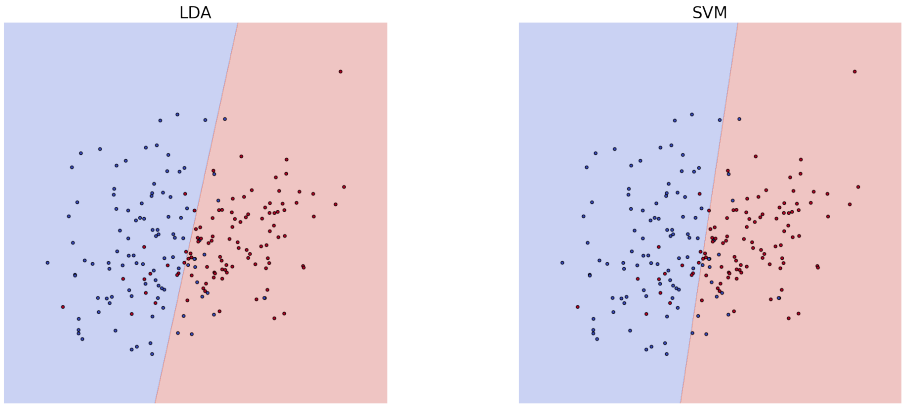


Fig. 3.9: A typical EMG processing pipeline, adapted from [243].

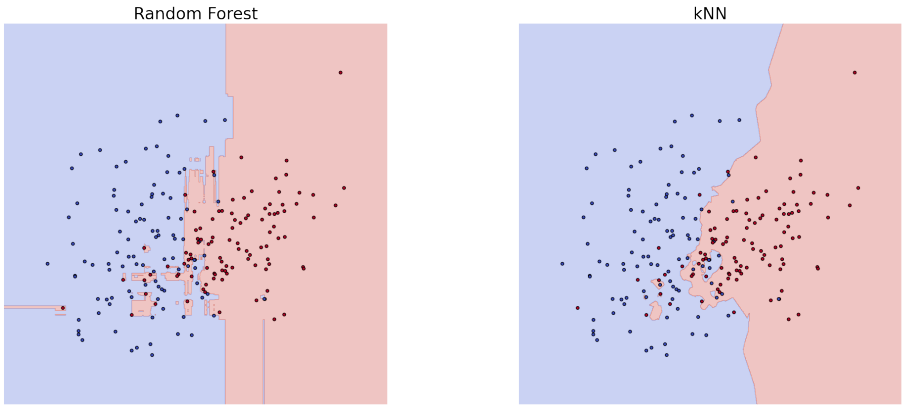
and they operate by reducing the feature vector to a set of features that is much lower in dimensionality but still highly representative of the initial information. Finally, classification may be performed by means of several options that are briefly discussed in the following sections.

3.3.1 Conventional Methods

Conventional methods for movement classification based on EMG signals rely on statistical algorithms like Linear Discriminant Analysis (LDA) [244], which projects data points onto lower-dimensional spaces to discriminate them from one class to another, or, more often, on machine learning algorithms, as evidenced by various studies in the field [245, 246]. Some examples are SVMs, K-Nearest Neighbor (kNN), and Random Forests [244, 247] which have occasionally been shown to outperform statistical methods, and have been recognized for their high accuracy in classification tasks. Fig. 3.10 exemplifies the differences in the classification planes between the mentioned methods for a 2-classes dummy dataset. The figures show how LDA and SVM tend to separate the data more neatly along a classification line (support, for the SVM), while the Random Forest and kNN algorithms find more diverse classification criteria. However, their practical deployment is often limited due to the variability in test conditions and the significant computational requirements needed to achieve such accuracy [11, 244]. In recent years, deep learning techniques have emerged as a promising alternative to traditional machine learning methods [248]. These techniques have demonstrated an enhanced ability to generalize to unseen conditions, potentially overcoming some of the limitations posed by conventional methods [249]. Despite these advantages, deep learning approaches remain computationally intensive, posing challenges for their application in wearable solutions, where computational resources are typically constrained. Notable examples of research in this area include the works of



(a) (b)



(c) (d)

Fig. 3.10: Examples of classification planes in a 2-class dummy dataset, for LDA 3.10a, (linear) SVM 3.10b, Random Forest 3.10c, and kNN 3.10d.

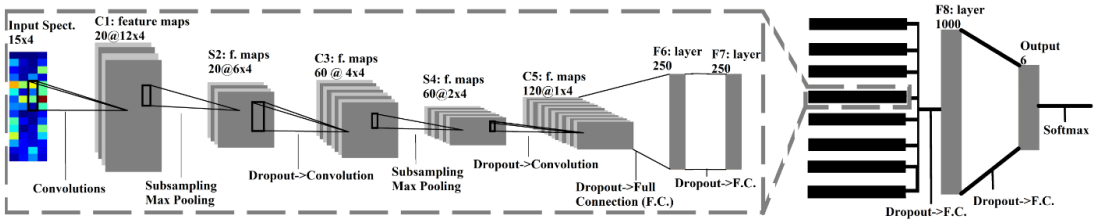


Fig. 3.11: Pipeline of the stage 2 processing of sEMG spectral data in [251]. The data is first parsed by a CNN with each channel individually. The outcome is then used for classification by two fully connected layers.

[250, 251], where frequency-based feature extraction systems such as the Fourier Transform (FT) and Wavelet Transform (WT) are employed before processing the data with a classifier. In particular, [250] compares the use of Fast Fourier Transform (FFT)-preprocessed data and data from the original EMG domain with different window size to train a CNN of varying size. They highlight that longer windows are preferable when employing FFT data and that overall, FFT data enables the CNN to achieve higher levels of accuracy, albeit significantly increasing the computational cost. In [251], the authors utilize the spectrograms of the EMG signals to train a 2-staged system, where the first stage determines whether the class is "Rest" or another action, and the second stage allows to identify the specific action, depending on the outcome of stage one. The second stage CNN is only utilized in case the outcome of the first stage classification is "Action". If this is the case, the spectrograms of each channel in the input are initially processed singularly, and finally collated into a sequence of two fully connected layers to perform the final classification. The schematics of this stage are reported in Fig. 3.11 for better understanding.

Additionally, a similar approach is seen in [252], where the authors integrate an attention-based system on top of the initial FT preprocessing stage. In particular, they utilize both time domain and frequency domain data to obtain intermediate classification results from two independent networks. These are then concatenated, as shown in Fig. 3.12, before being fed to a secondary classifier that produces the final classification output. This method illustrates the continuous evolution and refinement of techniques aimed at improving the accuracy and efficiency of movement classification based on EMG signals. Nonetheless, the computational cost of implementing this solution can be prohibitive. Has discussed in 3.2.2, Transformers-based solutions can become rather costly to run in terms of memory requirements and computational power requirements. In the reviewed work, three of them are present. On top of this, the FT applied to the data can also prove to be a costly operation [250], thus further increasing the

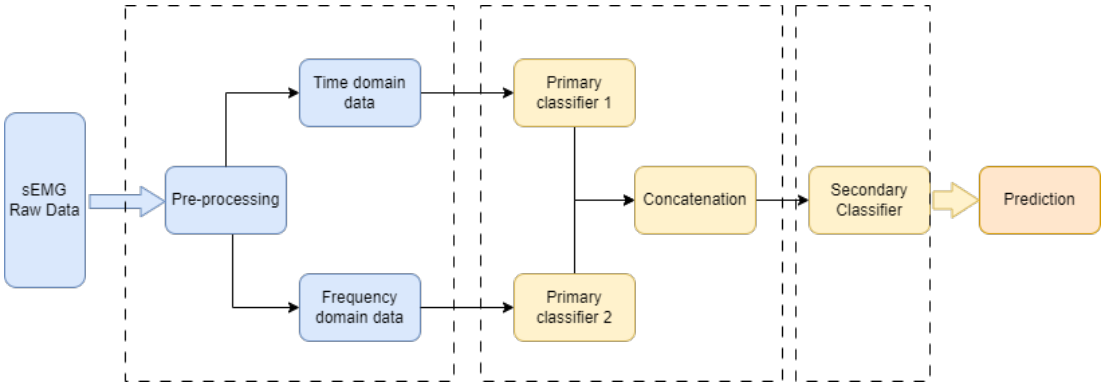


Fig. 3.12: Schematics of the classification pipeline presented in [252]. The pipeline has been adapted from the original work, and shows the progression of the operations performed starting from the raw data pre-processing and moving onto a two-staged classification based on transformers.

power requirements of this solution.

In summary, while both machine learning and deep learning methods show considerable promise for EMG-based movement classification, their deployment in practical, wearable applications remains a significant challenge due to computational constraints and variability in real-world conditions.

3.3.2 Neuromorphic Computing

The operational limitations of myoelectric prosthetics include the necessity for timely information processing and low power consumption, which implies a low computational budget [10]. Wearable devices have limited power and memory constraints, and often cannot have direct access to CPUs or GPUs [253]. As discussed, an sEMG setup is composed of a series of electrodes that sense muscle movements. The more the electrodes, the higher the resolution, the higher the amount of generated data. Such data can require significant memory to be stored and could incur losses if this is not possible, thus causing disruptions. The lack of access to performing hardware often means that data needs to be transmitted to some computing platform for processing, thus increasing the overall latency. If this is not the case, solutions must be scaled down to fit onto the hardware constraints of the wearable device, with potential losses in performance and with significant costs on the autonomy of the device itself. Such limitations bring Neuromorphic computing systems forward as a potentially suitable alternative to conventional solutions. As a matter of fact, despite still being scarcely applied as highlighted by their absence in systematic review papers [249], approaches employing SNNs have been proposed for the classification of movements and gestures

starting from EMG signals. The work in [254] showcases how a relatively simple SNN can achieve notable accuracy levels whilst maintaining low power requirements and low latency. They compare two relatively simple SNNs (an SCNN and a fully connected SNN) to other methodologies in the literature using the Ninapro DB5 dataset. They encode EMG data using a delta encoder, which consists of emitting a positive or negative spike whenever the signal surpasses a threshold (or goes below its negative). In this way, for each channel, two additional channels are created, effectively transforming the 8-channel data into 16 spike trains per each window the signal was broken into. The fully connected SNN achieves better results than the SCNN, with an overall accuracy of 74%. This is also deployed on a Loihi chip, where the authors measure their models' latency and energy consumption as being considerably low. However, the gap in accuracy with other methods remains large. Another work [255] presents a slightly deeper SNN which is able to achieve higher levels of accuracy on an FPGA board. Regarding the data encoding procedure, they utilize the same as the previously reviewed work, but apply their encoding to the first and second derivative of the signal too, to emphasize the speed and acceleration of muscular movements. Their FPGA-implemented model not only achieves a higher accuracy (85.6%) than [254], but also consumes less power (1.708 mW as opposed to the 41 mW in the previous work). Nonetheless, despite the increase in accuracy, their solution employs a 500 ms-long window, which may pose challenges for a real-time implementation which normally requires an overall latency < 300 ms [256], and has been shown to incur repeatability (and thus generalization) issues [242]. Furthermore, it still lags behind conventional DL methods presented in earlier sections in terms of accuracy. In [257], the authors propose an approach that combines a delta encoder for the EMG data, a spiking version of RNNs for feature extraction based on excitatory and inhibitory neurons, and an STDP-trained classifier which is used to cluster inputs into distinct groups. In Fig. 3.13 the schematics of this are reported for an easier understanding as found in the original work. Their method is shown to achieve better performances than some of the references whilst maintaining low latency. The authors also estimate an extremely small average energy consumption (due to the neurons) of $1.155\mu W$. Nevertheless, this approach requires a pre-processing step to normalize and remove outliers in the data, which could render real-time processing more challenging. Furthermore, it employs a three-stage processing through a randomly connected RNN for feature extraction, an STDP-trained classifier layer, and a clustering algorithm for class determination. This could prove non-trivial to operate, and, due to the several operations performed

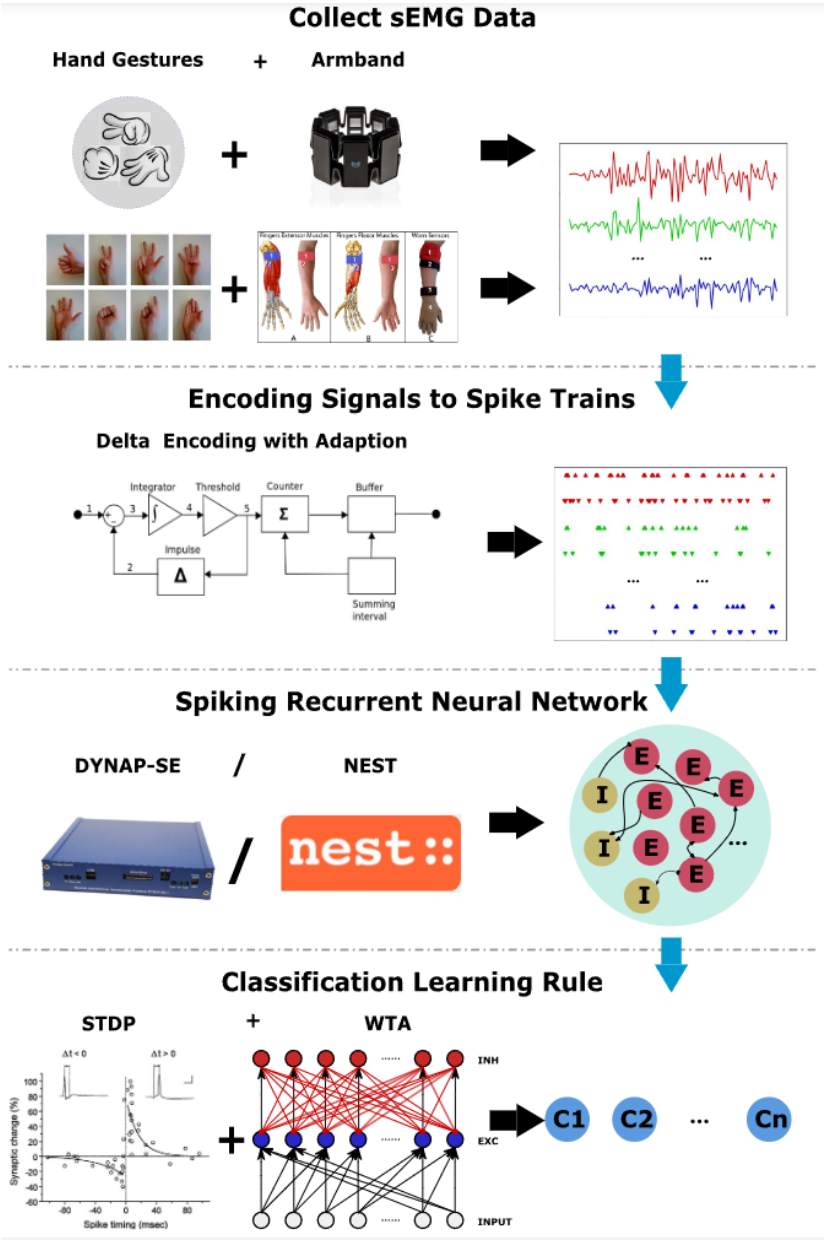


Fig. 3.13: Schematics of the pipeline adopted from [257]. The raw EMG data is firstly pre-processed to remove outliers and normalize it, then an RNN composed of excitatory and inhibitory spiking neurons extract information for an STDP-trained classifier. The output of this is then used to cluster the data into separate groups and thus perform classification.

outwith the SNNs, the overall system could potentially incur further processing times and computational requirements. Finally, the authors in [258] utilize an interesting approach based on the fusion of EMG data and event-camera data. They collect gesture data as shown in Fig. 3.14 (data collection setup), pre-process the EMG signals and parse the data using different network models. They design two types of SNNs, a spiking CNN and a spiking MLP, that are compared to conventional DL counterparts. Their solutions are also implemented and tested on neuromorphic hardware. Their solution demonstrates a 600x increase in efficiency with respect to more conventional systems and highlights an energy consumption on NM hardware in the order of magnitude of μJ. An interesting point worth considering is that, in their investigations, the use of EMG data alone always results in considerably poorer performance ($< 25\%$ accuracy). The use of gesture imagery always results in better performance, but the combination of the two is the solution that provides better accuracy overall. Nevertheless, while in the case of conventional DL it does not vary considerably, when relying on the sole use of EMG data the energy consumption of NM hardware is on average one order of magnitude lower as compared to imagery data. This is probably due to the much inferior amount of information that is propagated through the network, but it underlines how important it could be to be able to rely on the EMG data alone. Overall, they demonstrate that a fusion approach allows for reaching higher accuracy levels than single-mode-based classification (either EMG or imagery data) and that SNNs can attain higher performance levels at a much lower energy cost. However, the solution they present has some inherent limitations. Firstly, they employ feature extraction prior to processing the EMG signals which introduces a computational and temporal overhead due to the fact that this kind of computation is likely to be performed on a conventional CPU; secondly, the utilization of a fusion between the EMG sensors and the event-based camera limits the use cases, for instance, it would not be compatible for implementation on prosthetic devices for everyday use; finally, the use of both types of data considerably increases the power requirements due to the higher number of events being propagated on the NM devices, and considerations must be made regarding the feasibility of this with respect to the real-world implementations.

Overall, the NM community has begun to take steps towards approaching the problem of EMG-based classification of gestures where the advantages brought by NM substrates and information processing fit well with the operational limitations that applications based on these sensors can have. The literature has highlighted that SNN-based solutions offer promising results not only in terms of energy

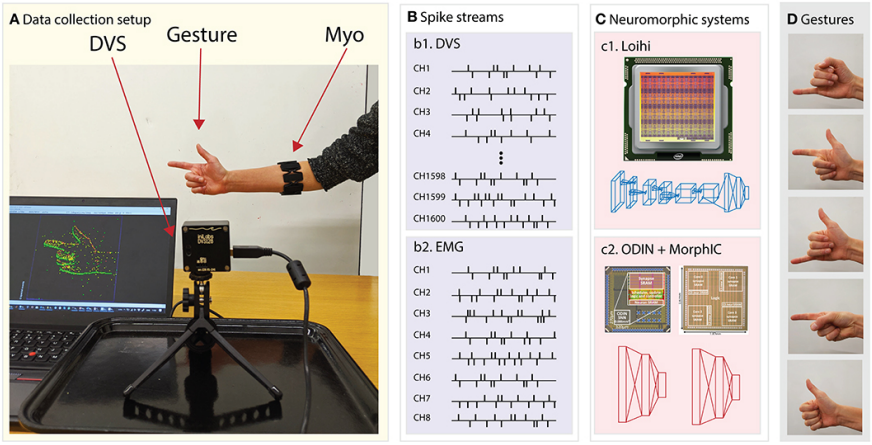


Fig. 3.14: EMG and event-camera fusion data collection, as reported in [258]. The gestures are performed while wearing a Myo armband and in front of an event-based camera. From (B), the number of channels related to the event-based camera is considerably higher (NxM pixels) than the 8 channels related to the Myo.

consumption but also in terms of attainable accuracy, thus strengthening the foundation for further research into this field. An interesting observation from this review of the literature is that the majority of the SNNs employed to tackle this problem have been trained using the SLAYER learning rule, in particular within the LAVA framework. A possible explanation for this regards the ability of SLAYER to learn delays between neurons as well as synaptic weights and in the possibility of deployment onto Loihi chips thanks to LAVA.

3.3.3 The Ninapro Dataset

The performance of the works reported above is normally evaluated using purposely built datasets. A popular one that is found across several works is the Ninapro [25] dataset. This dataset comprises several databases with different specifics but with the general objective of providing training and testing data for EMG gesture recognition. Particularly relevant for this thesis is Database 5 (DB5). This comprises data regarding 52 different gestures plus the rest condition for a total of 53 subdivided into three types of exercise (A, B, and C). An example for each exercise is reported in Fig. 3.15. The collected data relates to 10 different subjects that were asked to perform each action for 6 times each. EMG data was recorded by means of two MYO Armbands [259] (Thalmic labs) positioned on the forearm for a total of 16 sensors (channels in the dataset) [25]. The dataset is considered to be an extremely challenging one [254]. Due to the fact that participants were requested to go back to the resting position after each repetition, the dataset is extremely imbalanced, with more than 60% of the sam-

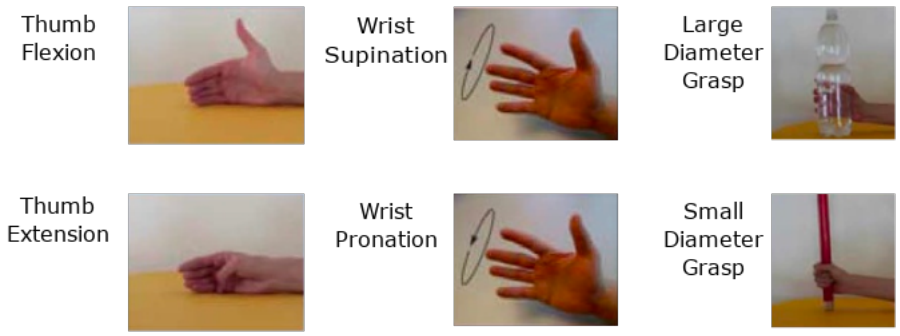


Fig. 3.15: Replicated example gesture images from the three different exercises in the Ninapro DB5 dataset.

ples belonging to the rest class. Fig. 3.16 reports a bar chart of the number of samples per each class in the validation set of Exercise A (as used in Chapter 6). Moreover, the remaining 52 classes have a relatively small number of samples. As such this dataset provides a good testing ground for the performance and robustness of proposed solutions, and is for this reason utilized in Chapter 6.

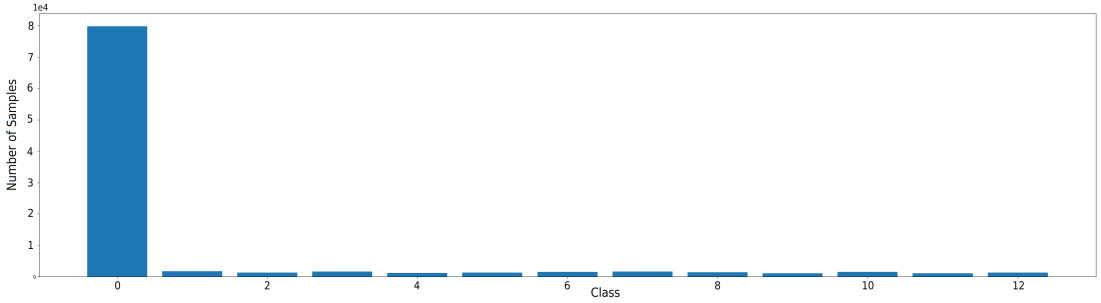


Fig. 3.16: Number of samples per class in the Ninapro DB5 (validation) dataset. The "rest" class ("0") is predominant.

3.4 Conclusions

This chapter has introduced two application domains that are of interest to this thesis. Time series forecasting is an ever-present problem across a number of different fields. The chapter introduces relevant background literature regarding conventional processing systems and highlights existing Neuromorphic approaches to this domain. From the literature, conventional approaches appear to be well established and able to achieve high levels of forecasting ability, but not without limitations. Conventional methods for forecasting often rely on statistical models, which include methods for accounting for sources of non-stationarity such as seasonality and trends in the data. These models become increasingly complex

to utilize and require high expertise in each different time series domain. Furthermore, they have been shown to fall short when compared to other DL-based solutions, especially when the data becomes increasingly complex. DL works offer promising results, at different costs. Conventional feedforward neural networks are not inherently capable of processing time-based information, such as the one contained in time series. As such, more complex structures like RNNs, LSTMs and Transformers are required. They promise high levels of accuracy but come with the cost of large memory requirements, large data sets, large training time, and large power consumption. One notable endeavor is represented by LLMs. LLMs make concurrently use of RNNs, LSTMs and attention-based mechanisms to process sequences of token in order to perform predictions and generate text-based responses. Their performance makes them an outstanding transformative technology, which, nonetheless comes at great environmental costs. In this landscape, Neuromorphic computing comes as an alternative both in terms of lower power requirements (in the view of an edge-device deployable time series forecasting solution) and in terms of time-based computational capabilities. SNNs have been discussed have having such capability thanks to spiking neurons and delay-learning, and the literature shows an interest in their application to this domain. Furthermore, despite many NM applications are mainly inclined towards relatively limited autonomous systems, an NM approach could also prove beneficial in cases similar to LLMs. Considered the scale and the amount of computations of LLMs, an NM solution could stand out as an alternative capable of achieving similar results whilst employing considerably less computations thanks to its spike-based logic that enhances sparsity and thus reduces the overall information propagation and power consumption. When approaching this problem with SNNs, however, time series are often required to be encoded before being processed. Nevertheless, the encoding is often only regarded as a means to transform information into spikes, rather than an opportunity to remodel the data into a more amenable form for learning. This poses the basis for further research in this direction, as in order to conceive an end-to-end NM process elements like conventional pre-processing of information cannot be included, or at least not in a form that requires conventional CPU computations. The chapter thus introduces two energy load and demand forecasting used later on in chapter 5. The two datasets, Panama and ETTh1, feature useful characteristics for the purpose of the work carried out later on, and represent and impactful way in which time series forecasting can be applied. Both datasets, in fact, feature different types of non stationarity (trends, periodicity, etc.) and are thus helpful in demon-

strating the effectiveness of the algorithms presented in chapter 5. Concerning EMG gesture classification, classical processing pipelines are presented, alongside deep learning solutions, similar to the time series case. Again similarly to the time series forecasting, conventional methods have shown promising results, both showcasing the advantages of employing frequency-based feature extraction pre-processing techniques, and the DL ability to generalize well to unseen data. This comes at the cost of the need for large datasets and ample computational resources, as well as high power requirements. Neuromorphic (NM) approaches in this area are also presented and reviewed with particular attention to those employing Ninapro DB5, a well-known challenging EMG gesture classification dataset, as a benchmarking dataset. The presented solutions demonstrate the feasibility of using SNNs to resolve this task but not without limitations. The review highlights in fact that this kind of work often falls short in either accuracy levels or real-time processing capabilities. Many works show considerable savings in terms of power consumption, once again underlining the potential of NM solutions, but can seldom compare with conventional deep learning approaches, or need to resort to pre-processing, convoluted architectures or extra types of input to reach acceptable levels of performance With the literature reviewed above, navigating through the following chapters will result in a smoother effort, as well as understanding the gaps that the presented contributions will try to fill.

Chapter 4

Simple and Complex Spiking Neurons

4.1 Introduction

Building a good knowledge base for the field of NM computing requires understanding the different components and tools that can be used to build a system. In the particular case of Machine Learning applied to NM, these range from Neural Network architectures to learning rules, spiking neuron models and tools to develop them. In Section 2.2.1 it has been shown that a multitude of different neuron models have been theorized in the literature [12]. Different neuron models vary in mathematical complexity, in the meaning of their parameters and in the neuronal dynamics that they expose. Nevertheless, most of the time in practical implementations only the simplest neurons are used, regardless of the evidence on their limitations that are present in the literature, with the LIF [70] neuron model being the de-facto standard choice. A review of the literature also highlighted the lack of studies on the differences in using different spiking neuron models within Machine Learning systems in the NM field. Indeed, a justification for the use of a neuron model with respect to its performance on a task is hardly found. From a hardware point of view, the LIF has been shown to be more efficient and less memory hungry to implement than multi-compartment models and the HH model [260], thus making it an understandable first-choice when researching NM solutions. However, the limitations discussed earlier have also been highlighted, with more complex models being able to attain a plethora of observed neural behaviours. But, even outwith hardware contexts, the reasoning behind the LIF choice is only given with regard to the model's efficiency or biological plausibility, however, no mention of the representational abilities of a model is made. While

this is still a plausible reasoning, it can be a limiting factor for the possible directions that both algorithmic and hardware NM research can take. Furthermore, the lack in the exploration of what functional differences could exist in different spiking neuron models make it not trivial to make a solid choice for a neuron model when developing an SNN. Within some specific contexts, the choice is constrained by the available hardware. Several neuromorphic chips allow to only adopt the specific neuron model that the chip is able to emulate [182, 261–264]. Until recently, the only chip allowing the implementation of customized neuron models was SpiNNaker [145]. However, the recently released Loihi 2 [101] chip adds to the list of chips with programmable neuron models, hence highlighting the importance of an accurate investigation on this matter.

This chapter presents a study aimed at finding whether the usage of diverse and more complex neuron models benefits learning with STDP in SNNs. The study simulates an extremely simple real-time scenario, where the SNN processes an NM input one time-step at a time and performs classification. This is enforced both in the learning stage and in the inference stage, hence ensuring that learning via STDP takes place in a way such that computations at each time step are dependent on the previous updates even within the same sample, thus being more biologically plausible [126]. A subset of single-variable spiking neurons with differing mathematical complexities was selected for comparison. To evaluate the above, each neuron model was employed in the same SNN and learning setting, and a number of experiments were designed to ensure robustness in the results. Section 2.2.3.1 reported how STDP can be susceptible to the presentation order of the data. In order to assess whether this effect could be amplified by the usage of any specific neuron model, the study also comprises a set of experiments specifically designed to analyse this possibility. The rest of the chapter is organized as follows: section 4.2 describes the event data utilised in the study; Section 4.3 briefly details the studied spiking neurons models; Section 4.4 describes the neural network and learning paradigm; Section 4.5 and Section 4.6 report a brief description of the classification an optimization methodologies respectively; section 4.7 reports details on the results of the experiments; section 4.8 highlights some key implications over the usage of neurons and that stem from the obtained results; chapter 4.9 summarizes the work, underlining the most important findings and outlining further interesting research directions.

4.2 Event-based Data

For the purpose of this study the N-MNIST [191], and the DVS Gestures dataset [192] are used to create binary classification tasks using couples of classes every time. The two natively neuromorphic datasets above are selected to assess the performance of the SCNN, while other non-native NM datasets are purposedly discarded as they do not possess a temporal domain, nor data is originally event-based. This is because tasks based on non-temporal have been shown to be possible to solve without the use of spiking neurons [18], whereas when the data contains temporal information, spiking neurons do play an important role, thus highlighting the need for such type of data to evaluate their capabilities.

Data in the N-MNIST dataset is collected by recording MNIST digits shown on a screen using a moving DVS camera. Specifically, the camera makes the same 3 predefined movements for every sample, each lasting roughly 100 ms. In this way, although the dataset is built on top of a non-neuromorphic one, data samples in the dataset are natively event-based, rather than being converted from a static image. Fig. 4.1 reports the count of events per each 5 ms time step throughout all the classes of the dataset. By contrast, data samples in the DVS Gestures dataset (see Fig. 4.2 for the inter-class distribution of events over time) are recorded using a fixed DVS camera in front of which participants move their arms according to instructions. Thus, 11 different classes of gestures are obtained, including for example arm rotation, waving or performing air guitar. The 11th class encodes "Other" random movements and is not considered in these experiments. This is because the overall objective of the study is to determine whether any difference is found in the model's performance when using different neuron types, rather than finding the best solution to a problem. It is thus of interest to create a relatively simple set of tasks where such a difference can more easily be noted, if present.

Event data comes in the form of Address Event Representation (AER)-encoded [265] files in which every sample is constituted by a sequence of events. Events are characterized by the specific time at which they occurred, by the location on the 2D plane and by the polarity (negative or positive light change). Similarly to [266], to make data usable by a 2D CNN, a 2D image is populated using all the events that took place between time t and $t + dt$, allowing at most 1 event per (x, y) coordinates. For simplicity, all events are considered to be positive and a batch size of 1 is used. As a result, the network processes only a single event

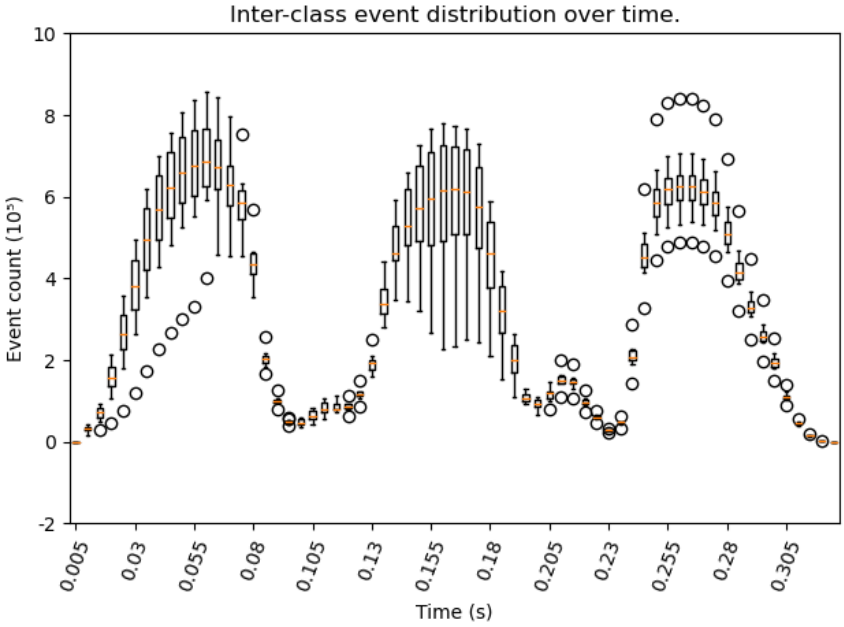


Fig. 4.1: Visualization of the number of events over time. The figure reports the collective mean and variation of events throughout all the classes. As can be seen, events tend to appear always within the same time ranges for all the samples of all the classes in the dataset, thus highlighting the lack of temporal significance. Values on the y-axis are scaled by a factor of 10^5.

Tab. 4.1. Table of hand-tuned parameters used for neurons. Each column represents a different parameter, as outlined in Section 2.2.1.2. The capacitance C is used instead of the resistance R by leveraging the equality $R = \tau_{rc}/C$. For every neuron, the time-step size used was 0.02, and the voltage threshold was recalculated for every 100 samples. Where no unit of measure is reported, parameters are considered dimensionless

Neurons	τ_m **(s)**	u_{rest}	C	Δ_T	Θ_{rh}	a	u_c
LIF	0.2	0	0.1	-	-	-	-
EIF	0.2	0	0.1	1352	216	-	-
QIF	0.2	0	0.1	-	-	0.01	216

image with a time resolution dt belonging to only one sample at a time. Finally, each event, or spike, is characterized with a value of $\frac{1}{t_s}$ in line with [12]. This is done in order to preserve the amount of charge that a spike carries regardless of the time-step (t_s) size. No further pre-processing is applied to the data.

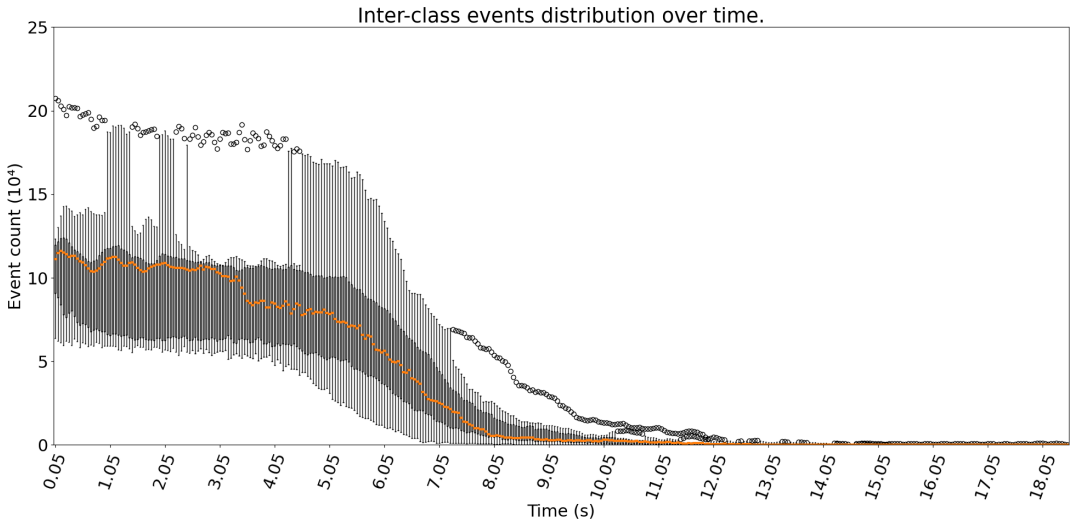


Fig. 4.2: Visualization of the number of events over time in the DVS Gestures dataset. The events span across the whole time domain and are highly dense for about the first 7 seconds. At that point, they start to decrease in number, however, the tail of the curve continues for a long time. This is due to some of the gestures lasting longer than others.

4.3 Spiking Neurons

The phenomenological family of neuron models is arguably the preferred option when developing spiking neural networks, as described in Section 2.2.1.2. This work specifically concentrates on three integrate and fire neurons, namely the LIF, the QIF and the EIF. At the core of this selection lays the LIF model. The LIF is the most widely used neuron model that embeds a time dependency through the membrane potential leakage. As such, it is selected as being the starting point of the comparison and drafts the requirements for the other neuron's selection. The models need to be point neuron models, real-valued and single-variable in order to ensure a fair comparison. The other models, the QIF and the EIF, align with these requirements being point-neuron models, real-valued (no complex numbers), and employing one single variable for computations. As a matter of fact, their constituent dynamics equations solely depend on the value of the membrane potential, but they all employ different types of dependencies from it. Furthermore, they are the base on which other multi-variable and popular neuron models are built, respectively Izhikevich's [16] neuron and the AdEx neuron model [12]. Finally, these two models have been reported to having better abilities to fit observed cortical neurons [12, 89]. The parameters used for these models in the experiments where common parameters across the three are kept the same (see

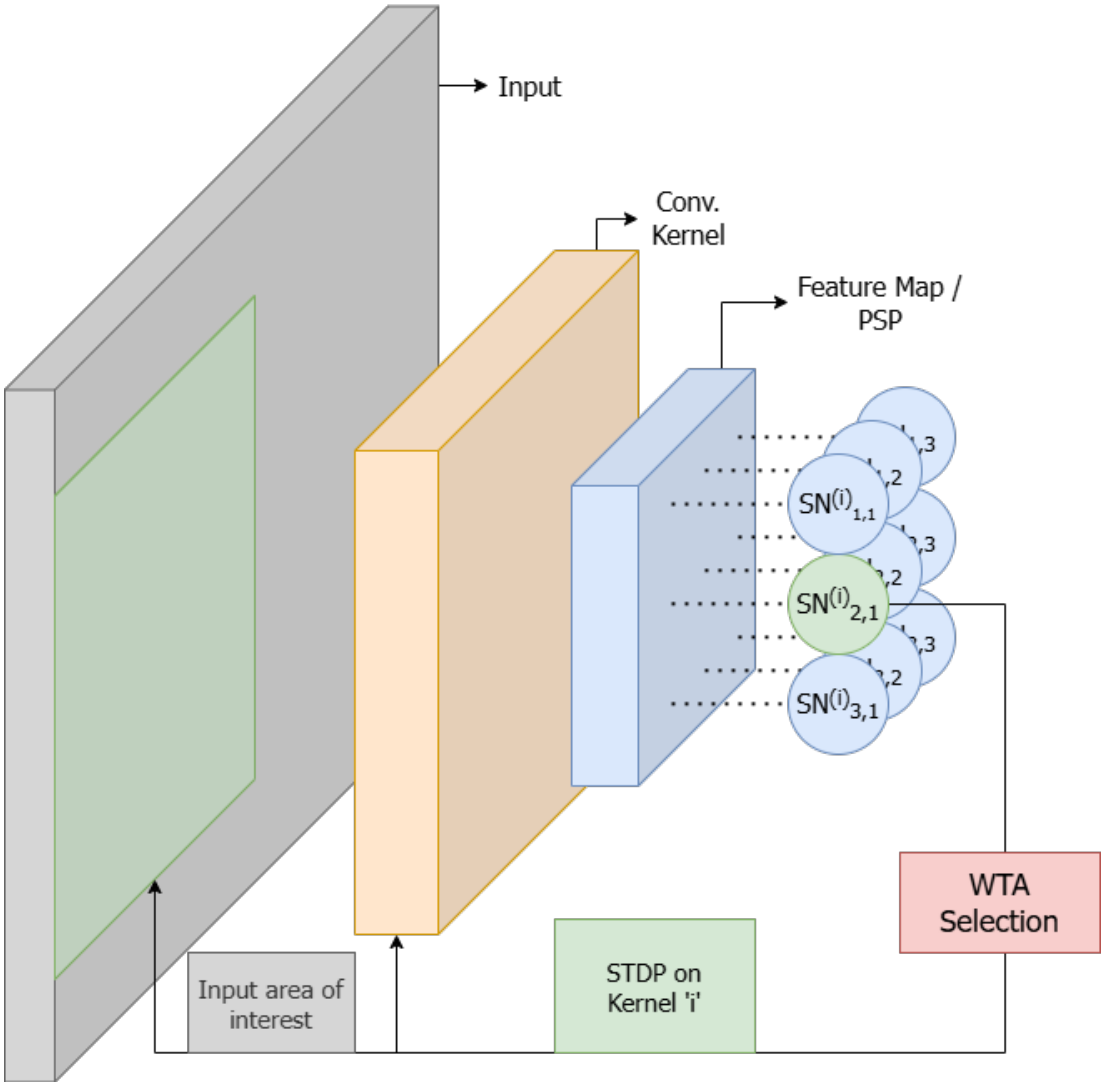


Fig. 4.3: Diagram of the Learning Pipeline. A 2D convolution layer parses the input spike map and produces N feature maps with width size WxH. Each value in each feature map is fed to a distinct neuron and the one spiking earliest is chosen as a winner by the WTA mechanism. STDP weight updates are then applied to the convolution kernel corresponding to that neuron.

Section 4.7) are reported in Tab. 4.1. From a practical point of view, experiments using such spiking neurons have been designed and carried out within a version of the SpykeTorch framework that has been modified and expanded to accommodate the needs of the study. Details about these implementations are reported in Appendix A as they can prove useful as a reference for future studies.

4.4 SCNN and Learning

When designing an ML pipeline including a neural network, many different factors could determine the outcome of the learning. The interest of this study is to assess the performance of an SCNN trained via STDP when one of its components, namely the spiking neuron model, is varied. Since the performance could be affected by a number of other components, a simple design for the experiment is chosen which involves the minor number of structural elements possible. The idea is to reduce the number of components that might impact the overall system performance, and to thus be able to draw stronger conclusions with respect to the role of the different spiking neuron models employed. To this extent, a simple single-layer convolutional network is employed where spiking neurons are embedded subsequently to the convolution operation on the input. This layer parses the input events and serves as a connection with following layer of spiking neurons. Fig. 4.3 illustrates the pipeline for a better understanding. In this setting, the weights of the kernel of the convolutional layer can be thought of as being synapse strengths, and the resulting feature map is the input post-synaptic potential to a set of spiking neurons arranged accordingly. In other words, the convolution represents the connectivity scheme for the spiking neurons. The use of a convolutional layer to link inputs with the spiking neurons allows these to more easily learn spatial features. The weights of the convolutional layer are the only parameters being learnt by the network and no pooling nor normalization is applied.

Learning in these experiments is carried out via the STDP rule presented and used in [134, 267]. This STDP rule follows the mathematical formulations given by Eq. (2.31), which is essentially a system of two parabolic equations that are applied depending on whether LTP or LTD needs to take place. One of the consequences of using this learning rule is that weight updates are self-regularized. In fact, the closer the weights get to the boundary values, the smaller the updates will be, allowing the weights to be refined more granularly as the learning proceeds. Another aspect that is important to highlight is in how this learning rule is applied. While the original theorization of Hebbian rules such as STDP states that the weight update should be proportional in value and sign on the time-difference between the post- and the pre-synaptic spikes, in a software implementation some approximations are required. Therefore, for each time step of the execution, LTP is applied on weights connected to input locations where there has been a spike, and LTD in all the others. In order to promote competi-

tive and differentiated feature learning, a kWTA (with k=1) learning paradigm is also employed. WTA allows only k neurons per time-step to be eligible for STDP updates, specifically, the ones firing sooner.

Finally, as a homeostatic mechanism to allow neurons to keep on firing despite the changes in their synaptic weights, individual neurons' thresholds are periodically re-calculated according to Eq.(4.1). This form of adaptive thresholding is often required when employing STDP, and is a common practice for SNNs [267].

$$V_{thresh} = \lambda \cdot R \cdot A \cdot \frac{t_s}{\tau_m} \cdot \overline{W} \cdot (W_k \cdot H_k) \cdot n_c, \tag{4.1}$$

and since $\tau_m = R \cdot C$, it is possible to equivalently re-write:

$$V_{thresh} = \lambda \cdot A \cdot \frac{t_s}{C} \cdot \overline{W} \cdot (W_k \cdot H_k \cdot N_c), \tag{4.2}$$

where A is the amplitude of the spike, in this case assigned to be $A = \frac{1}{t_s}$, R is the resistance, C is the capacity, τ_m is the membrane time constant, $\overline{W}$ is the average value of the synaptic weights, W_k, H_k and N_c are the width, height and depth (number of channels) of the synaptic kernel, and λ is a regularization parameter that takes values in the range $[0, 1]$. In Eq.(4.2), the term $A \cdot \frac{t_s}{C} \cdot \overline{W}$, can be explained as being the average effect perceived on the membrane potential as a result of a single spike, whereas the second term, $(W_k \cdot H_k) \cdot n_c$, scales this effect to the size of the synaptic kernel. Therefore, Eq.(4.2) calculates what would be the average post-synaptic potential perceived in the case the input was dense with spikes. The parameter λ serves as a regulation of what percentage of this amount would be necessary to reach before emitting a spike.

4.5 Classification

Because an unsupervised learning rule is employed to enable learning, labels are not used at any point during the learning of the weights. However, labels are needed to classify data samples, therefore a system similar to [59, 268] is put in place that counts, for each neuron, the number of times it spiked in response to samples having a given label. At the end of the training phase, each neuron is assigned the label for which it spiked the most during training.
During the inference phase, for each data sample, a sequence of spikes is collected and weighted depending on the order they arrive. More specifically, the train of output spikes is stored for each output class. When the last time step computa-

Tab. 4.2. Table of optimized parameters used for neurons. Each column represents a different parameter, as outlined in Section 2.2.1. The capacitance C is used instead of the resistance R by leveraging the equality $R = \frac{\tau_{rc}}{C}$. For every neuron, the time-step size used was 0.02, and the voltage threshold was recalculated every 100 samples. HC stands for Hand Clapping, while RHW stands for Right Hand Wave. Where no unit of measure is reported, parameters are considered dimensionless

Neurons	0 vs 1							HC vs RHW						
	τ_m(s)	u_{rest}	C	Δ_T	Θ_{rh}	a	u_c	τ_m(s)	u_{rest}	C	Δ_T	Θ_{rh}	a	u_c
LIF	0.0602	0	0.2983	-	-	-	-	0.1435	0	0.3020	-	-	-	-
EIF	0.2578	0	0.2178	32	91.14	-	-	0.2389	0	0.2086	32	69.42	-	-
QIF	0.2804	0	0.2178	-	-	0.001	69.72	0.0621	0	0.1026	-	-	0.0393	275.95

tions are completed, for each class, a counter is increased by an amount that is the inverse of each spike's arrival time, thus promoting earlier spike times. For example, if the output neuron corresponding to class X has produced spikes at times 2,3, and 4, the counter for class X will be increased by 1/2, 1/3 and 1/4 for a total of 1.08. A label is finally assigned depending on the highest value.

4.6 Hyper-parameter Optimization

Defining a good set of parameters for a machine learning system is a non-trivial task. This often requires a lot of expertise and hand tuning and is greatly error-prone. To reduce the possibility of selecting sub-optimal parameters for neurons which would result in poor performance, an optimization system is thus used to find reasonably good parameters for the experiments. Specifically, the BOHB optimization [269] is adopted using the HpBandSter library. This technique combines Bayesian Optimization (BO) and Hyperband (HB), a resource allocation and early stopping strategy. To use BOHB, the implementation of the SCNN pipeline had to be adapted so that it could be optimized using the HpBandSter library. More importantly, domains in which every parameter was allowed to vary had to be defined. For example, the time-constant τ_m could be drawn from the interval $[0.06, 0.26]$. Moreover, because the optimization process could be task-specific, separate optimizations for each different task were performed. The final parameters obtained are summarized in Tab. 4.2. As a result of this process, more robust conclusions about the performance of the neurons can be drawn, as a poor configuration of hyper-parameters is more unlikely to have been selected.

4.7 Results

The SCNN is trained as outlined in Section 4.4 using a subset of the N-MNIST and the DVS Gestures datasets. In order to maintain the task simple, 4 distinct couples of classes from each dataset are randomly selected, and, for each, a separate binary classification task is defined. This ensures a higher degree of generalisation of the results than the case of testing on one couple of classes only, while also helping reduce the possibility that some results only depend on a particular choice of coupling. This simplification is functional to the objective of the study, which is to determine whether any change in the mathematical complexity of the neurons can impact the overall performance. It is thus not in the scope of this study to find a solution that works well on difficult tasks, but rather to present a simple case where such difference can be noted. To a certain extent, by presenting a simpler problem, the overall experiment garners a higher degree of generalization: if a difference is in fact notable, it could be amplified in more complex tasks. Furthermore, the order in which data is presented to the SCNN might influence a system employing STDP as a learning rule. This is due to the fact that STDP rewards and builds on the inputs that are presented earlier. Therefore, to ensure independence from this behaviour of STDP, every experiment is repeated a total of 11 times per task.

4.7.1 Same Hyper-Parameters Training

Experiments are first conducted using hand-tuned hyper-parameters. These were found by a trial-and-error practice and represent a set of parameters that enabled learning for the task at hand. This means that neurons using these parameters were able to emit spikes and to have the weights adjusted in a way that enabled the learning of representations of the inputs. Where possible, the same hyper-parameters are adopted for all the neurons in all the experiments on each dataset. Since the QIF and the EIF models introduce two different hyper-parameters each, each of these hyperparameters undergoes a further hand-tuning. Results of the training sessions are shown in Tab. 4.3 for the N-MNIST-based tasks and in Tab. 4.4 for the DVS Gesture-based tasks. Here, for every task and neuron model, the average and best test accuracies achieved are reported , calculated according to Eq.(4.3):

$$accuracy = \frac{TP + TN}{TP + TN + FT + FN}, \tag{4.3}$$

Tab. 4.3. Table of results on the N-MNIST dataset. In each cell, the mean accuracy ± the standard deviation values are followed by the best accuracy found (after the comma). Values are rounded to the closest second decimal value.

Neuron Model	0 vs 1	2 vs 9	3 vs 7	4 vs 8
LIF	0.77±0.11, 0.93	**0.74±0.06, 0.79**	**0.73±0.04, 0.78**	**0.57±0.02,** 0.59
EIF	**0.80±0.12, 0.94**	0.64±0.07, 0.71	0.65±0.04, 0.71	0.55±0.04, **0.61**
QIF	0.57±0.03, 0.61	0.54±0.03, 0.58	0.51±0.02, 0.55	0.52±0.02, 0.55

Tab. 4.4. Table of results on the DVS Gestures dataset. In each cell, the mean accuracy ± the standard deviation values are followed by the best accuracy found (after the comma). Values are rounded to the closest second decimal value. In the table, HC stands for Hand Clapping, RHW for Right Hand Wave, RACW for Right Arm Clockwise, AG for Air Guitar, RACCW for Right Arm Counter Clockwise, AR for Arm Roll, LACW for Left Arm Clockwise and AD for Air Drums.

Neuron Model	HC vs RHW	RACW vs AG	RHCW vs AR	LACW vs AD
LIF	0.53±0.06, 0.60	0.50±0.05, 0.56	0.54±0.13, 0.66	0.52±0.09, 0.69
EIF	**0.58±0.12, 0.77**	**0.50±0.04, 0.58**	0.53±0.11, 0.66	0.46±0.05, 0.52
QIF	0.57±0.10, 0.71	0.48±0.04, 0.52	**0.62±0.06, 0.67**	**0.53±0.09, 0.75**

where TP stands for true positive, TN for true negative, FT for false true and FN for false negative. On each column (task), the best average score and the absolute best score are highlighted in bold.

By examining the results on the N-MNIST-based tasks in Tab. 4.3, the LIF neuron model is found to perform better than the other two on average. Indeed, the EIF has higher average accuracy only on the 0 vs 1 task, whereas the QIF model fails to achieve accuracy levels high enough to match any of the other two counterparts.

Considering the results reported in Tab. 4.4 concerning the DVS Gestures dataset, the situation differs slightly. The performance of both the LIF and the EIF neuron models, on average, decreases drastically, whereas the QIF maintains similar levels of accuracy as in the N-MNIST case. Nevertheless, both the EIF and QIF demonstrate superior classification abilities throughout and their top accuracy levels often surpass those of the LIF model. These trends in the accuracy levels highlight two main aspects. Firstly, the complexity of the N-MNIST

Tab. 4.5. Table of results using optimized hyper-parameters. In each cell, the mean accuracy ± the standard deviation values are followed by the best accuracy found (after the comma). Values are rounded to the closest second decimal value. Optimization and evaluation is performed on one representative task per dataset only.

Neuron Model	0 vs 1	HC vs RHW
LIF	**0.95±0.02,** 0.982	0.53±0.05, 0.625
EIF	0.74±0.15, 0.93	**0.57±0.11, 0.67**
QIF	0.90±0.05, **0.985**	0.55±0.05, 0.625

data is lower than that of the DVS Gestures dataset. Indeed, in both cases, the same neural network architecture (with the exception of the kernel size, to account for the higher dimensionality in the Gestures data) and neuron models were employed, yet the accuracy in the DVS Gestures is on average considerably lower, hence highlighting the greater difficulty of the task. Data samples in the DVS Gestures dataset arguably have richer visual and temporal features which render the learning more difficult when compared to the N-MNIST. Secondly, the richer temporal diversity of the features might be better represented by means of neurons with richer voltage dynamics, such as the QIF and EIF. As shwon by the experiments, in fact, these two are steadily better than the LIF models and even though in some instances one performs more poorly, the other still attains higher accuracy, possibly as a result of a better affinity to the temporal dynamics found in that particular task.

4.7.2 Optimized Hyper-Parameters Training

As a second set of experiments, the optimization system outlined in Section 4.6 is employed to obtain a set of hyper-parameters that is heuristically optimal for a specific scenario. The optimization is carried out on the "0 vs 1" and on the "HC vs RHW" tasks for each neuron model individually. A total of 24 optimization iterations is allowed for each neuron and task, to not favor any experiment over the others. Results are reported in Tab. 4.5. Once again, after obtaining the optimized hyper-parameters, each model is trained and evaluated a total of 11 times to increase the robustness of the results.
Since the optimization is task-specific, the models are only evaluated on the two representative tasks they were optimized on. In the case of the "0 vs 1" task

based on the N-MNIST, overall, the accuracy levels drastically increase. The LIF model accuracy grows by nearly 20 percentage points on average; however, the most striking increase is the accuracy of the QIF model, which gains 33 percentage points on average and 37.5 in the best case. This not only highlights the sensitivity of neuron models to their hyper-parameters, but also confirms that neurons with more complex dynamics can perform just as well as simpler ones.
In the case of the "HC vs RHW" task, instead, it is possible to see a slightly different trend. In the first place, the results are surprisingly worse than those obtained through hand-picked parameters. The hypothesis is that this is because the optimization system required more iterations to find a good set of hyper-parameters. As stated above, the same number of optimization iterations as in the case of the N-MNIST dataset was allowed to maintain consistency. However, if the objective was to attain higher accuracy levels, a greater number of iterations would have been beneficial given the higher level of complexity in DVS Gestures features. Secondly, although still struggling to achieve higher accuracy levels, the SNNs employing QIF and EIF averagely outperform those with the LIF. This confirms the results obtained using the same hyper-parameters and strengthens the hypothesis that richer dynamics can be beneficial when employed on data with a richer set of temporal features.
Another point worth considering is the variability of the results obtained. Spanning from the N-MNIST-based tasks to the Gestures-based ones, the different neuron models demonstrate accuracy levels with a standard deviation of up to 15 percentage points. The hypothesis is that this effect is caused by the order in which data is presented to the system in relation to the STDP learning rule which, as reported at the beginning of this section, is sensitive to such order. It is also possible to observe that the EIF model has higher fluctuations on average, which could possibly reflect a higher sensitivity to this effect.

4.7.3 Sensitivity to data presentation order

The previous paragraph hypothesised the EIF neuron model was particularly sensitive to the presentation order of the data as a result of being trained through STDP. Driven by this, in this section, a series of controlled experiments aimed at studying this hypothesis is performed.

In order to assess whether any neuron is more sensitive to the order of the data, some parameters of the experiment must be constrained to ensure the final results do not depend on these. Since a homogeneous set of neurons (they all share the same parameterization) has been used in the experiments, the only

Tab. 4.6. Table of standard deviations on the N-MNIST dataset. Each cell reports the standard deviation calculated across all the experiments per each neuron and each task. Values are rounded to the closest third decimal. For each task, the highest values are highlighted.

Neuron Model	0 vs 1	2 vs 9	3 vs 7	4 vs 8
LIF	**0.092**	0.063	**0.068**	**0.061**
EIF	0.054	**0.074**	0.056	0.024
QIF	0.076	0.069	0.065	0.021

varying elements are the convolutional weights (randomly initialized), and the order of the data itself. Hence, consistently with the previous methodology, a set of 11 experiments per task and neuron model is designed in which the initialization of weights is fixed using a seed for the random number generator. Further to this, each neuron model processes data presented in the same order. In other words, each neuron model is evaluated using the same order of data before changing to another order for a total of 11 times per task. By doing so, the possibility of any neuron model displaying certain sensitivity levels as a result of fortunate/unfortunate randomization of the data is ruled out.

As a measure of the network sensitivity to the presentation order of the data, the standard deviation of the accuracy across all the experiments per each neuron and task is considered. These are reported in Tab. 4.6 and Tab. 4.7. The average sensitivity with 95% confidence intervals for each neuron model is reported in Fig. 4.4.

A one-way analysis of variance (ANOVA) is performed to verify whether any neuron is significantly more sensitive than others. The test assumptions were checked. Levene's test was non-significant ($p = 0.256$), indicating that the assumption of homogeneity of variance was not violated. Normality was checked with a Q-Q Plot. No deviations were noted. No significant difference among the three neuron models in sensitivity to the order of the data as found, $F(2, 21) = 0.514$, $p = 0.605$, $\eta_p^2 = 0.047$. In the context of the experiments above, these findings indicate that using either neuron model does not increase nor decrease the sensitivity to the presentation order of the data.

Tab. 4.7. Table of standard deviations on the DVS Gestures dataset. Each cell reports the standard deviation calculated across all the experiments per each neuron and each task. Values are rounded to the closest third decimal. For each task, the highest values are highlighted in bold.

Neuron Model	HC vs RHW	RACW vs AG	RHCW vs AR	LACW vs AD
LIF	0.086	0.051	**0.122**	0.028
EIF	0.049	0.049	0.114	0.032
QIF	**0.088**	**0.079**	0.101	**0.074**

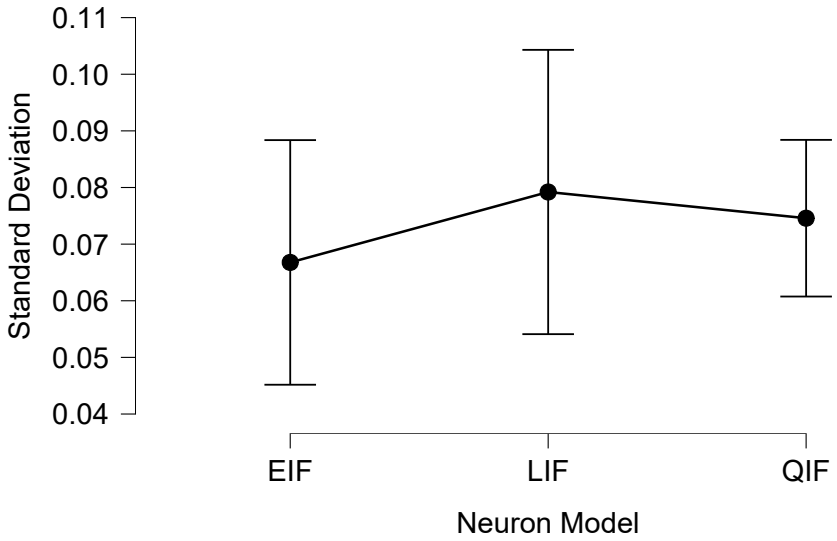


Fig. 4.4: Plot of average standard deviations with 95% confidence intervals. The average was computed across all tasks per each neuron.

4.8 Discussion

4.8.1 Implications of Using Different Neuron Models

The usage of spiking neuron models has some inherent implications on the machine learning pipeline from the implementation and the theoretical points of view.

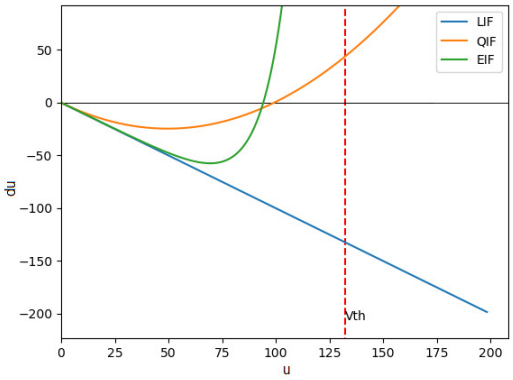
Concerning the implementation, spiking neurons come with a whole set of hyper-parameters to tune. Considering the LIF, the simplest version requires a single parameter (the time constant or a leakage term), but other implementations might include up to 5 different parameters, such as the refractory period or the time-step size. By using the QIF or the EIF, there are at least two new and non-optional parameters to consider (see 2.2.1).

Determining a good set of hyper-parameters is a non-trivial task [270]. Although

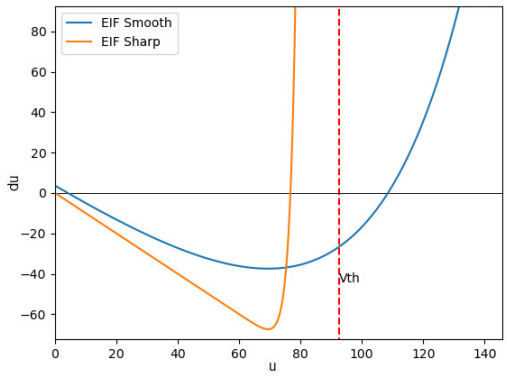
in the NM field a lot of inspiration is taken from the human brain, it is not possible to simply assume that the same parameters that work in such a complex system would still be applicable in a simplification such as an SNN. It is hence necessary to tweak the hyper-parameters to meet the needs of the use case or, alternatively, to define a parameter optimization strategy that does that heuristically in an automated way. However, the latter solution also often requires making guesses about the domain in which parameters can vary and it requires a long time to compute. Furthermore, hyper-parameters can be correlated in some way, thus making both the hand-tuning and the automated optimization process more difficult. As a result, from an implementation point of view, using neuron models that require more hyper-parameters can significantly increase the usage complexity.

From a theoretical point of view, using different neurons or varying the parameters means opening to different non-linear dynamics and excitability patterns. Fig. 4.5 and Fig. 4.6 provide a visual understanding of these differences. In the LIF, the membrane potential updates depend linearly on the previous state of the membrane potential itself. The EIF manifests a similar relationship up to certain values of membrane potential (Θ_{rh}), after which the relationship assumes a more non-linear (exponential) aspect. The QIF loses any linear relationship in favour of a quadratic one. These dynamics play a role in the excitability (regions) of a neuron [12]. For instance, an EIF with a smooth exponential term (blue line in Fig. 4.5b) will receive more mitigated updates throughout (slow-forgetting neuron), whereas an EIF with a sharp exponential term (orange line) will receive more negative updates up to the cutoff threshold Θ_{rh} and then highly positive ($+\infty$) ones, thus immediately reaching the firing threshold V_{th}. Hence, the second example would be a fast-forgetting neuron, but the cut-off threshold will act as an early firing threshold, as any subsequent update would bring the membrane potential up and above the actual firing threshold. A similar example is reported in Fig. 4.6, where a change in the time-constant τ_m makes a LIF neuron forget faster or slower. This in turn has effects on the excitability of the neuron and its firing pattern.

The different firing abilities discussed above need to be considered within the context of the application where the neurons are used. By considering the case of a homogeneous SCNN that is trained on a dataset in which the temporal distribution of events is similar for every sample, it might be pointless to have a wide range of excitability patterns as more complex neurons have. In fact, the increased amount of parameters would make it more difficult to find the right



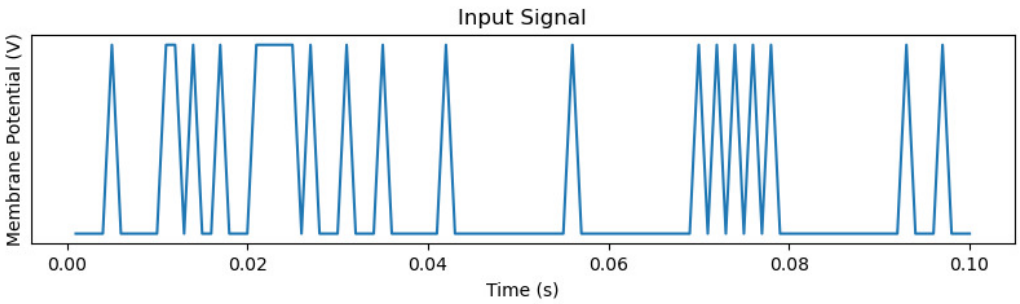
(a)



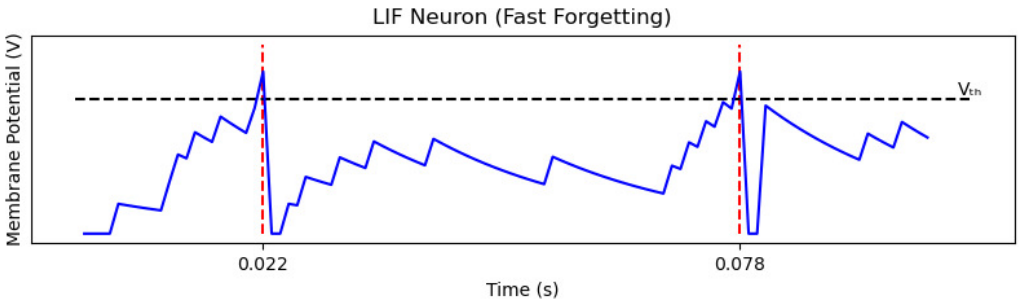
(b)

Fig. 4.5: Example of membrane potential dynamics of the spiking neuron models. In both figures, the y-axis represents the variation of the membrane potential du, while the x-axis represents the value of the membrane potential itself u. Fig. 4.5a compares the three spiking neurons, whereas Fig. 4.5b is an example of how varying the sharpness parameter Δ_T can affect the dynamics of the EIF.

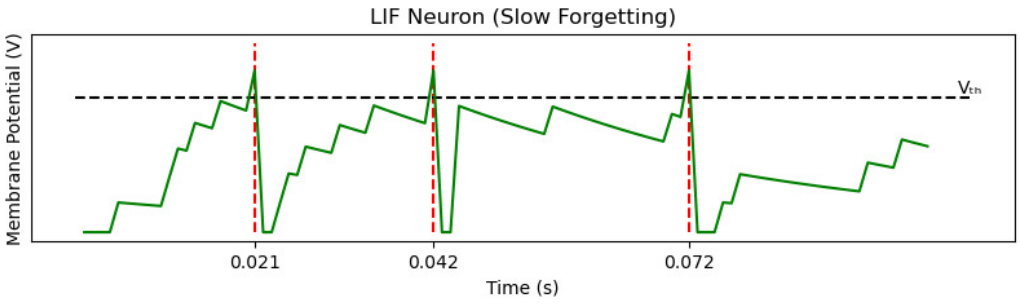
excitability that works well with that data. At the same time, they would likely come at a higher computational and power cost. Conversely, if the dataset is considerably diverse in terms of the temporal distribution of events, it would arguably prove useful to have a broad range of excitability patterns to choose from. Therefore, a heterogeneous network of spiking neurons would possibly be able to learn better or simply more features. In this context, employing more complex neurons with variable parameters can be significantly beneficial.



(a)



(b)



(c)

Fig. 4.6: Example of Changing Hyper-Parameters in a LIF Neuron. The "Fast forgetting" neuron (smaller time constant τ_m) (Fig. 4.6b) can only spike twice in response to the input (Fig. 4.6a). The "Slow forgetting" one (Fig. 4.6c) can fire three times and maintains a higher membrane potential throughout. Note that the "Slow forgetting" neuron also fires earlier, i.e. it requires less (close) spikes to reach the threshold. Both the neurons have a refractory period of 2 ms.

4.8.2 Temporal Features and Neuron Performance

In the discussed experiments, an extremely simple homogeneous SCNN was used to perform a simple classification task on a simple subset of the N-MNIST and DVS Gesture datasets. The N-MNIST dataset, although natively event-based, is not naturally dynamic. The original data, the MNIST handwritten digits, are static images that do not contain temporal dynamical features. As such, the temporal features that are instead present in the N-MNIST are crafted. Furthermore,

these dynamics are obtained by moving a DVS camera using the same sequence of movements with the same timing. Fig. 4.2 shows that, as a result, the distribution of events throughout each sample present in the dataset is roughly the same. This means that the temporal features are not different from one another and are hence not discriminative of different classes of samples. Indeed, as discussed in the previous paragraph, using a homogeneous SCNN was enough to achieve reasonably high accuracy levels, despite the lack of diversity in the dynamics of the embedded neurons.

Concerning the performance in such homogeneous settings with three single-variable neuron models, it has been found that all of them have the ability to perform well. The difference however is in the cost of using one neuron rather than the other. From the experiments, it has been found that when hand-tuning parameters, the LIF neuron achieved averagely high accuracy levels, with the EIF neuron being better at times. When using a set of optimized hyper-parameters, a considerable improvement in the overall classification accuracy was observed, with the QIF achieving a 98.5% accuracy in the best case, thus surpassing its counterparts. The same QIF model performed rather poorly when using non-optimized parameters. As mentioned in Section 4.7.2, this highlights the fact that, despite the data displaying simple spatio-temporal features, more complex neurons are still able to perform well. The cost of achieving such results can, however, become rather high.

When employing DVS Gesture data, the situation is slightly different. In this case, as depicted in Fig. 4.2, there is no recurring distribution of events across different classes. Instead, events are distributed throughout the whole time domain. Each class of gestures has a distribution of events that varies with respect to the others, even more, because of the fact that different actions require a different time to be executed. Thus, the temporal features in this dataset are more important and diverse. As a matter of fact, this is also shown by the results obtained using the same network as in the previous case. Here, although the performance gain is still modest, the EIF and QIF neurons steadily attain better classification accuracies than the LIF model. Since the same setting was used for all the experiments, this is likely traceable to the aforementioned differences in temporal features, which are now more diverse and complex than those in the N-MNIST dataset.

4.8.3 Temporal Features and Depth of the Network

The matter of temporal variety in the features being better represented by more complex dynamics opens up further questions as to their use in SNNs. Indeed, when considering a hierarchical NN, the deepest layers normally learn more abstract representations, whereas the early layers typically learn to distinguish simple patterns, such as edges or corners [35]. This is easily conceivable when thinking about spatial features. For example, when a set of lines is recognized in the early layers of a CNN, these could later be understood to be a square, and further down the network as a house. Although it can be more difficult to imagine, when the time dimension is included, similar scenarios can arise where features relate temporally rather than only spatially. It thus seems straightforward that the temporal relationships might vary in complexity in different stages of the network depending on the task at hand.
It has been shown that the use of more complex neuron models improves the performance on more complex tasks at the level of one layer. When combining several of these layers in a hierarchical network more uses of their non-linear dynamics could arise, as they would combine several spatio-temporal features built up in previous stages to understand compound featural patterns.

4.9 Conclusions

In this chapter, a simple unsupervised SCNN has been considered and the effect on the overall performance of changing the underlying neuron models analysed. To achieve this from a practical point of view, the framework extension developed and discussed in Appendix A has been used. Firstly, a set of 4 binary classification tasks has been defined using 4 couples of classes from the N-MNIST dataset on which the SCNN has been repeatedly trained and evaluated. Experimental results on these tasks show that all three neuron models (LIF, QIF, EIF) can achieve top-level accuracies, albeit the more complex ones require more fine-tuning. In a second instance, the DVS Gestures dataset is instead utilised, which exposes a richer set of features from both the visual and temporal points of view. In this case, the EIF and QIF steadily outperform the LIF on all 4 tasks drawn from this dataset. Further to this, an analysis of the sensitivity to the order of presentation of the data of the SCNN with each neuron model has been performed. The analysis shows that none of them implies a statistically relevant difference in terms of sensitivity and that such sensitivity is probably only relatable to the use of STDP, in these experiments. While the scenarios analyzed in this study

focus on specific neuron models, the findings are applicable to a broader range of neurons. This study demonstrates that variations in mathematical complexity can influence performance levels and that these variations are closely tied to the complexity of the dataset's features. These insights pave the way for further research into determining which neuron models are best suited for specific functions. Additionally, the results highlight the potential performance benefits of selecting an appropriate spiking neuron model. Furthermore, it highlights that further research aimed at unveiling the role of the dynamics of neuron models in deep hierarchical learning would be highly beneficial to close the gap between conventional DL approaches and SNNs. These findings can also be generalized beyond the scope of binary classification. If a difference is in fact notable when solving a task as simple as binary classification, the simplest type of classification, when extending to a greater number of classes such difference could become even more evident, given the need for the separation ability of the network to be greater, and for higher chances of fine-grain cross-class features in the data to be similar. In general, the combined use of spiking neurons and STDP to build an SNN can lead to solutions that are more amenable to edge computing devices. If an SNN with accurately-selected spiking neurons is deployed on edge-devices, STDP training can be employed directly on the device thanks to the considerably lower amount of computation and memory required for it to operate. This is because STDP can operate weight updates in real-time and in a localized fashion, thus not requiring computationally-intensive gradient calculations and large sets of memory to store intermediate backpropagation results.Other future studies could consist of analysing further relationships between the neuron models and other components of the learning pipeline, such as the neural network architecture, and the learning rule.

Chapter 5

Approaching Time Series Forecasting via Derivative Spike Encoding and Spiking Neural Networks

5.1 Introduction

When considering the application domains for SNNs, the literature is rich with works focusing on resolving tasks like tracking, object detection, classification, and optical flow, to name a few [200]. Typically the end goal of these applications is to develop an algorithm that could be deployed onto an edge system with limited capabilities (hence requiring low Size, Weight, and Power – SWaP), which, as largely discussed in the literature [10], can be achieved by leveraging the NM substrates that could host such algorithms; equally one may want to leverage the high dynamic range of a NM camera that allows it to see rapidly-moving objects better than a conventional vision sensor; or yet again one could inspect a scene where the background is relatively static, and where, potentially, an NM approach could therefore drastically reduce the memory requirements. To this extent, researchers make use of a plethora of different datasets as a means to validate a given NM algorithm. However, that of understanding what makes a good dataset to prove an SNN's worth is still a very active field of research [18, 271] and wide consensus is yet to be reached. At the same time, the research community is always active in trying to determine potential applications where employing NM-based systems could prove to be beneficial when compared to more conventional methods [10].

Time series forecasting is an ever-present task of interest in various research fields. Some examples are finance, meteorology, and logistics, but the list is long [211, 214, 220, 226]. When broken down into its core components, other types of data like videos or NM recordings can also be regarded as being time series, as they are but a collection of time-indexed images or events. However, when referring to time series, the most common connotation is that of time-indexed data points that are not normally associated with the visual domain.

As discussed in Chapter 3, traditional methods for time series forecasting include ARIMA, ETS, and STL[200]. These methods are based on mathematical models that capture the statistical properties of the time series data, such as trends, seasonality, and residual errors. While these methods have been widely used for many years, they can falter in accurately capturing complex non-linear dependencies in the data [272]. More recent methods include Wavelet-based and Fourier transform-based ones that can capture more complex properties in the data [209, 210].

With the recent advancements in DL, deep neural networks have become a popular approach for time series forecasting. Methods such as LSTM networks and RNN have shown promising results on a range of time series datasets, thus proving deep neural networks to be a viable solution in this domain. Such models have been further outperformed by the advent of the Transformers and, more in general, large language models. These groundbreaking solutions have established a strong state of the art in several fields, among which time series forecasting, but at the cost of even more intense computational costs, as outlined in Chapter 3. However, these require utilising complex systems to be able to process data in the time dimension, so that such methods are often unfeasible for real-time or online deployment [217, 218].

NM computing operates on the principles of event-based information processing, which is thought to be more biologically plausible and energy-efficient than traditional, rate-based neural networks [17]. This has been rendered possible by the technological advancements in NM vision sensors [160], and NM chips [101, 145, 182, 261], which enable sparse, asynchronous perception and processing. Such sparsity and asynchronicity can be fully exploited by means of Spiking Neural Networks, however, in the context of time series forecasting, their inherent time-based nature is what puts them forward as a potential alternative to the current approaches in the literature. In fact, thanks to the combined action of the learnt synaptic weights and complex internal dynamics of spiking neurons, SNNs have the capability to create internal representations (and thus, extract)

of complex temporal features in the data [17]. As such, they could be employed to resolve this kind of task not only efficiently, but also effectively. As a matter of fact, the performance gap between conventional DL algorithms and NM-based ones has been a critical aspect limiting the use of NM systems. However, recent research [196, 273, 274] is increasingly demonstrating how that gap can be narrowed, thus bringing the SNNs close to being on par with their ANN counterparts, if not surpassing them on certain tasks.

Chapter 3 discussed how time series can be essentially classified into stationary and non-stationary [200]. Non-stationary time series can potentially lead to misleading inferences of poor performances if not adequately dealt with. As such, many time series analysis and prediction models rely on the assumption that the time series is stationary or take actions to remove components that make a time series non-stationary (e.g. trends, seasonality). One such way that is considered to be effective as a general approach to transforming non-stationary series into stationary ones is differencing [200]. Through differencing, the effects of trends and seasonality can be smoothened and the time series made more stationary. Because of this, it is often employed prior to applying models like ARIMA, or ETS (see Chapter 3).

The main research question that this chapter aims to answer is whether it is possible to devise an NM encoding system that embeds this concept in such a way that an SNN can learn from it to perform predictions. The working principle behind an NM vision sensor is affine to the concept of differencing: an event is only emitted whenever a difference in lighting is above a certain threshold [160]. Therefore, it seems reasonable to devise a perception system that not only translates real values into spikes, but that does so in a way that potentially increases the forecasting capabilities that an SNN can achieve. This further translates into understanding whether it is possible to enhance this learning by means of ad-hoc designed loss functions that leverage biologically inspired concepts and the spike representation obtained from the encoding system. By finding an answer to the questions above, the aim is to contribute to developing time series forecasting as a potential main direction of research, where the use of SNNs could prove a valid alternative to conventional systems; or, on the reverse, where time series forecasting could be used as a benchmark task to assess the performance of an SNN model in general. In this sense, an investigation is carried out, covering some of the main points required to construct a time series forecasting paradigm with SNNs, starting from the encoding of the data into spikes to the design of the learning scheme. The motivations behind these investigations are dual. Firstly,

from an algorithmic point of view the innate ability of SNNs to process time-based information can be a crucial element that can benefit forecasting tasks in time series. If it is possible to engineer SNNs to leverage this core principle for time series, the gains in performance could be outstanding. Secondly, from a hardware-based point of view, existing solutions in DL can be extremely costly to operate. In Chapter 3, it has been shown how tasks like sequence analysis and forecasting solved by LLMs can be viewed as a special case of time-series. In such a view, the design of SNN-based solutions can not only improve performance as per the previous point, but also contribute to shifting towards more sustainable applications.

To perform this exploration, two electricity load forecasting datasets are used. Time series data is normally recorded by means of conventional sensors, that report values in the $\mathbb{R}$ domain of real numbers as a continuous stream. To transform the dataset into a format that is more amenable to an SNN, a novel encoding algorithm inspired by the differencing system and by dynamic vision sensors is proposed, as discussed above. A relatively simple SNN is built and is trained by means of the SLAYER [24] learning rule, so as to verify its ability to learn from such data. As a means to further improve the learning ability of the network, two novel loss functions are also presented that are inspired by the biological concept of inter-spike interval [12], and by the knowledge of the meaning of each spike from the encoding. The exploration begins by analysing the effects of the differencing encoding. To do this the encoding parameter space is manually sampled in a manner that is suitable for the dataset at hand, the signal is reconstructed starting from the obtained spike encoding, and the error is quantified. A more detailed analysis shows how, given the right conditions, the representation generated through the encoding system approximates the derivative of a signal. In order to observe whether the designed pipeline is able to perform forecasting on a sinusoid signal, a set of experiments is performed where the decoded output is compared with the derivative of the input signal. Following this, another series of experiments with varying hyper-parameters and loss functions are carried out on both datasets, aimed at gathering a number of results that can be utilized for an in-depth and robust analysis of the proposed system.

The rest of the chapter is organized as follows. Section 5.2 elaborates on the details of the devised system, from the encoding to the developed loss functions. Section 5.3 Describes the experimental pipeline and reports on the results obtained. Section 5.4 contains insights into the analysis of the obtained results, the implications and the limitations of the presented methodology. Finally, Section

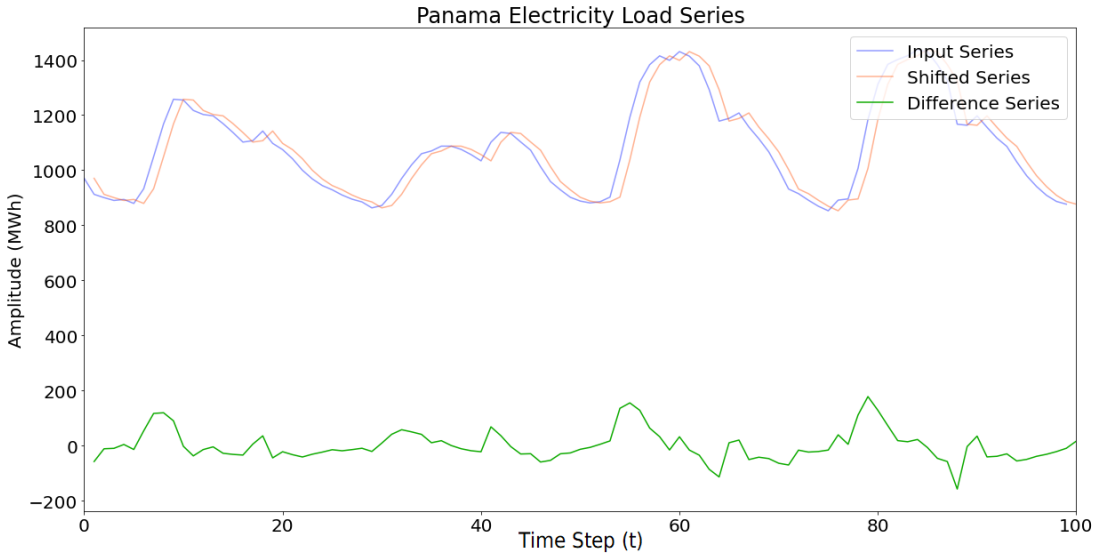


Fig. 5.1: Excerpt from the Panama dataset. The blue and orange lines are the signal and the lag-1 version of the signal respectively. The green line is the difference signal.

5.5 draw a summary of the study, highlighting key findings and possible future directions stemming from this piece of research.

5.2 Data Encoding and Processing

To approach the time series forecasting problem two datasets are primarily used, the Panama [21] and the ETTh1 [239] datasets. Both these datasets contain energy readings coupled with other information, such as temperature at the time of the reading and wind speed. Peak electricity loads in Panama present a high level of variability across different days in a month, as testified by Fig. 5.2, while the ETTh1 is a largely used electricity load forecasting dataset that exhibits considerable non-stationarity [23]. In this work, the focus is on a univariate forecasting problem; therefore, only the electricity load readings are considered and used as the input variable and the target variable. Fig. 5.1 reports an excerpt of the electricity load from the Panama dataset. The datasets above were collected using conventional (non-neuromorphic) sensors; therefore, they are composed of real-valued data points. Because of this, to conceive an end-to-end NM system, there is the need to transform the data into a spike-based representation that can be processed by an SNN. Therefore, this study defines an encoding mechanism that draws inspiration from NM vision sensors [160] as a way to enact a differencing step, while transforming the data into spikes emitted by a population of neurons. NM vision sensors produce a positive or negative event at pixel level whenever

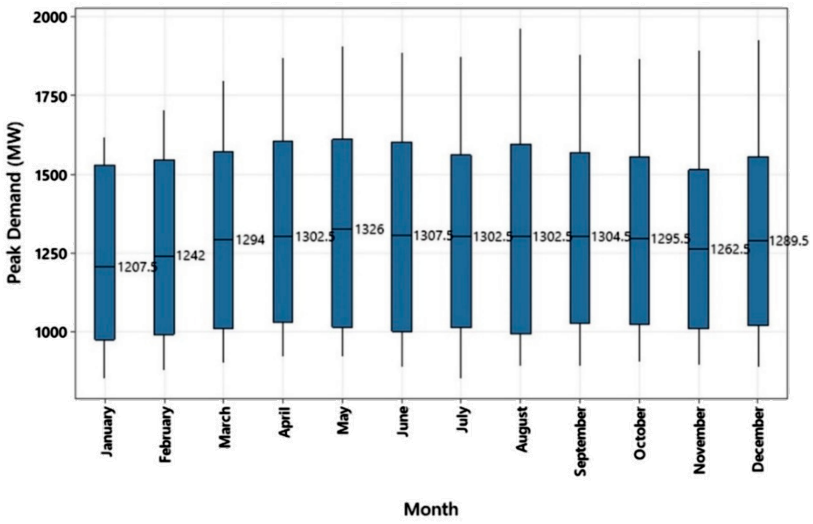


Fig. 5.2: Peak electricity demand across different months in Panama. Adapted from [22].

there is a light intensity change in the scene (positive for low-to-high, negative for high-to-low). The change needs to be strong enough, i.e. above a chosen threshold, for the camera to consider it as such and thus emit an event. As a result, the NM vision sensors perform thresholding on the input, increasing sparsity in the representation, reducing noise propagation and omitting unchanging, i.e. redundant, elements. Drawing inspiration from this, the encoding mechanism transforms a real-valued input into a sequence of spikes emitted by a population of neurons. Each neuron in such a population is responsible for emitting a spike whenever the change in the signal is above a certain threshold $V_{th}^{(i)}$, both in a positive and negative direction. Neurons can be characterized by the following model (here assuming a positive threshold V_{th}):

$$\begin{aligned} v[t] &= x[t] - v[t-1] \\ s[t] &= \begin{cases} 1 & \text{if} \quad v[t] \geq V_{th} \\ 0 & \text{otherwise} \end{cases} \\ v[t] &= x[t], \end{aligned} \tag{5.1}$$

where $v[t]$ is the state of the encoding neuron at time t, $x[t]$ is the value assumed by the time series at time t, and $s[t]$ is the output spike train at time t. In other words, if the input signal has a strong enough variation from time step t to time step $t+1$, one of the neurons will emit a spike; if no change happens, or if the change is too small, no spike will be emitted, hence encoding a differenced version

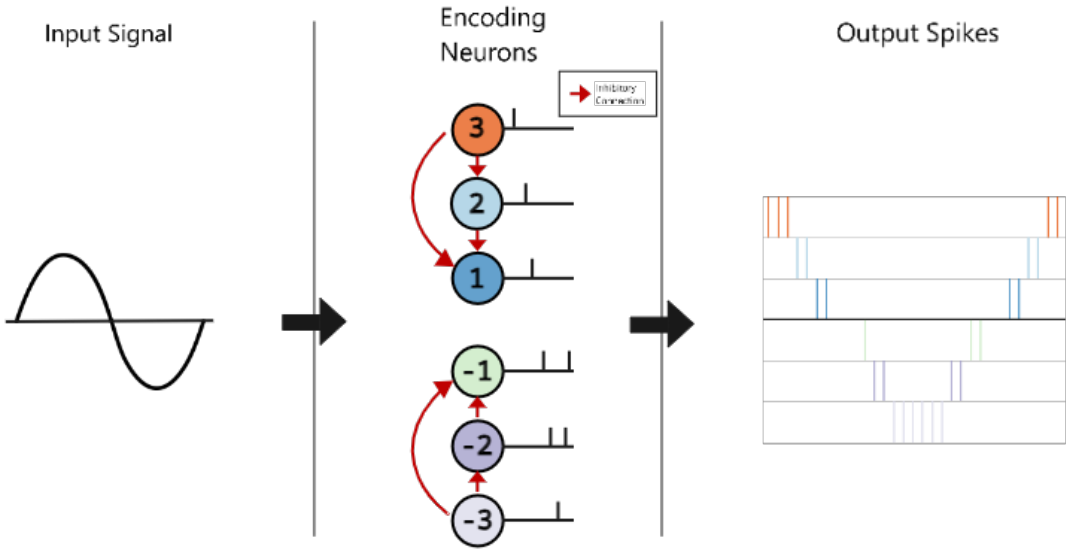


Fig. 5.3: Schematics of the encoding paradigm. The input is concurrently parsed by all the encoding neurons, which fire upon seeing a certain change. Inhibitory lateral connections in the populations ensure that if a change triggers neurons with higher thresholds, the other ones are prevented from spiking. The numbers in the encoding neurons represent the threshold multipliers relative to each neuron, rather than the threshold itself. If the base threshold were 0.5, they would have thresholds 1.5, 1, 0.5, -0.5, -1 and -1.5.

of the signal whilst increasing sparsity and reducing noise, similarly to an NM camera. Fig. 5.3 depicts a schematic example of the aforementioned encoding system. The input is parsed by the population of encoding neurons and a spike train is produced as an output. During the encoding, neurons associated with higher thresholds implicitly inhibit those corresponding to lower thresholds. In terms of simulation using the datasets above, this is achieved by considering the difference of the input signal with a delayed version of itself by one time step. A representation of this can be seen in Fig. 5.1 (green line).

The so-obtained encoding can be reversed by considering the threshold each neuron is associated with. Similarly to a quantization problem, the fidelity of the reconstruction of the original signal from the encoding is proportional to the granularity (number of neurons and value of thresholds) of the encoding. In order to quantify the information loss from the encoding, Tab. 5.1 reports an exploratory search of the reconstruction MSE on the Panama dataset using a different number of encoding neurons with different thresholds. Different sets of multiplicative thresholds are used for the encoding, i.e. each neuron is assigned a value $n \in \mathbb{Z} \setminus \{0\}$, then multiplied by the base threshold, so that the actual threshold would be $V_{th}^{(i)} = n \cdot V_{th}$. The selection of the base thresholds is performed

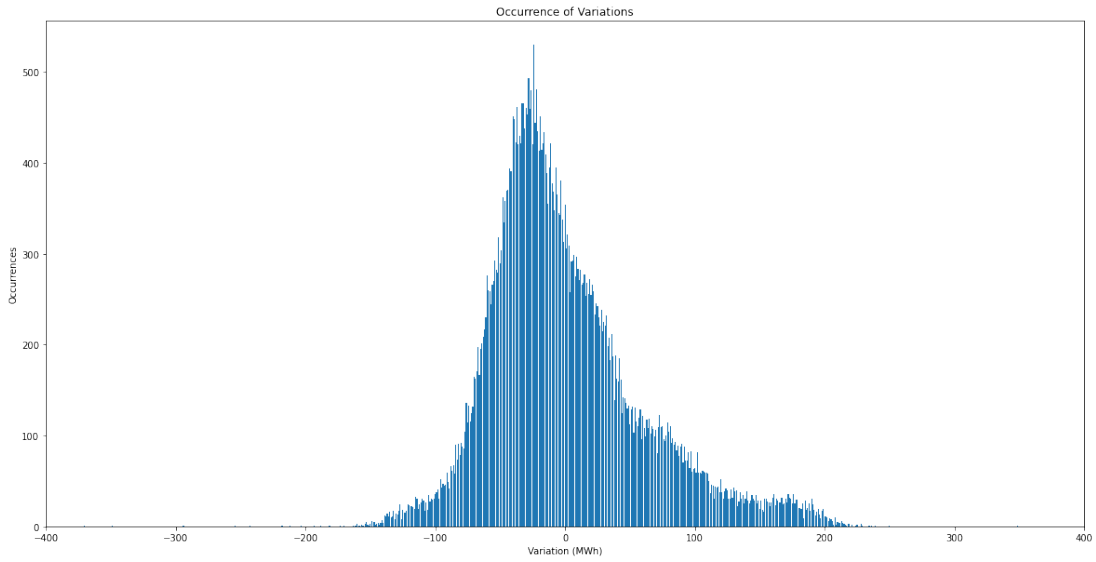


Fig. 5.4: Bar chart of the value changes in the Panama dataset. Each bar represents the number of times that change in value is found in the data from time t to time t+1 (i.e. $\Delta x = x(t+1) - x(t)$). Note that several of the changes are zero or close to zero, hence potentially not requiring any information propagation depending on the choice of thresholds in the encoding layer.

manually and is aided by the analysis of the value changes present in the dataset. By visualising the count of each value change from time t to time $t+1$ (e.g. how many times the time series has had a change of 20 MWh at one time?) as reported in Fig. 5.4, it is possible to estimate the ranges the threshold could be varied in. In the case of the Panama dataset, there is a higher number of negative variations, but they hardly surpass the $\pm$ 100 MWh. Interestingly, the minimum MSE is not obtained by using the largest number of neurons and lowest threshold, thus highlighting how threshold selection can be a crucial step. Fig. 5.5 shows an excerpt using the parameters relative to the minimum MSE value obtained. As it can be observed, the reconstructed signal follows rather faithfully the original signal, except from it incurring in some information loss for certain ranges of values.

5.2.1 Approximation of the Derivative

In its essence, the encoding system considers the average variation in amplitude of an input signal between two points in time, such that:

$$m = \frac{x(t + \Delta t) - x(t)}{\Delta t} \tag{5.2}$$

Tab. 5.1. Mean Squared Error between the original signal and reconstructed signal using different encodings. Neuron multipliers should be interpreted as referring to two neurons each (positive and negative versions of each). For instance, (1,2) with base threshold 9 refers to neurons with thresholds (-18, -9, 9, 18). The range(1, 60, step=2) indicates a range of values starting from 1 and increasing by two up to 60. In bold, are the best MSE values from the reconstructions.

Neuron Multipliers	**Base Threshold**				
	9	**13**	**23**	**33**	**53**
(1)	2852.88	2578.11	2033.40	1690.30	1476.04
(1, 2)	2259.65	1846.16	1187.73	910.15	944.43
(1, 2, 3)	1788.29	1328.03	731.51	576.37	814.01
(1, 2, 3, 4)	1417.18	962.62	478.96	433.10	800.42
(1, 2, 3, 4, 5)	1126.00	704.65	337.55	380.05	791.72
(1, 2, 3, 4, 5, 10)	494.50	**274.81**	**311.93**	354.15	773.62
(1, 2, 3, 4, 5, 10, 20, 30)	347.37	**274.67**	**283.07**	338.37	767.10
range(1, 60, step=2)	509.33	938.30	2530.13	4449.87	6885.43

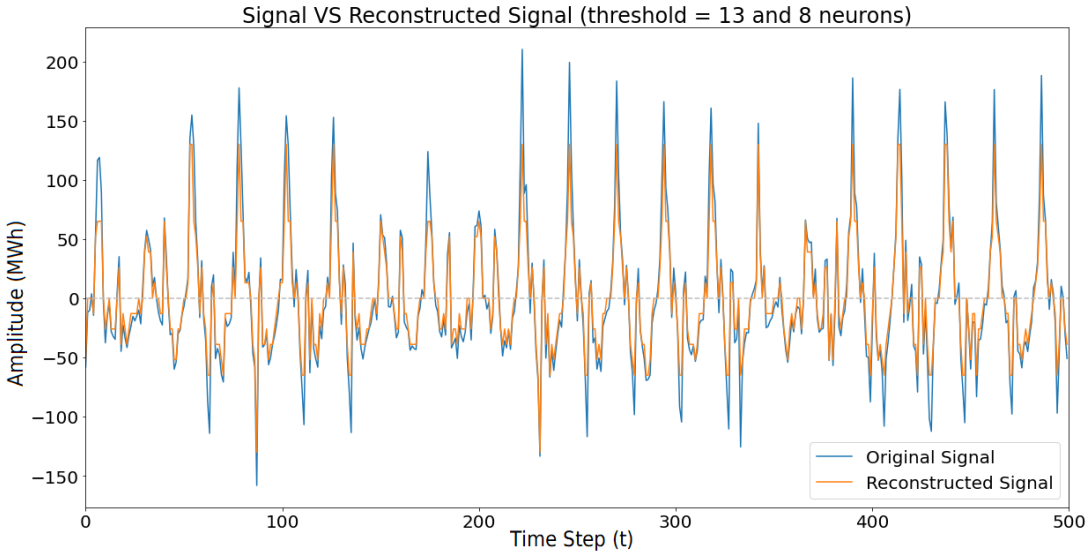


Fig. 5.5: Extract from the Panama data after differencing versus its reconstructed version.

where $x(t)$ is the input signal and Δt is the chosen time step. It is possible to observe that Eq. (5.2) denotes the difference quotient, or the slope, of signal x around time t. Interestingly, as Δt becomes smaller, Eq. (5.2) becomes an increasingly better approximation of the derivative of the signal over time as in Eq. (5.3):

$$\frac{d}{dt}x(t) \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}. \tag{5.3}$$

Thus, assuming that $x(t)$ is smooth around time t, the encoding method

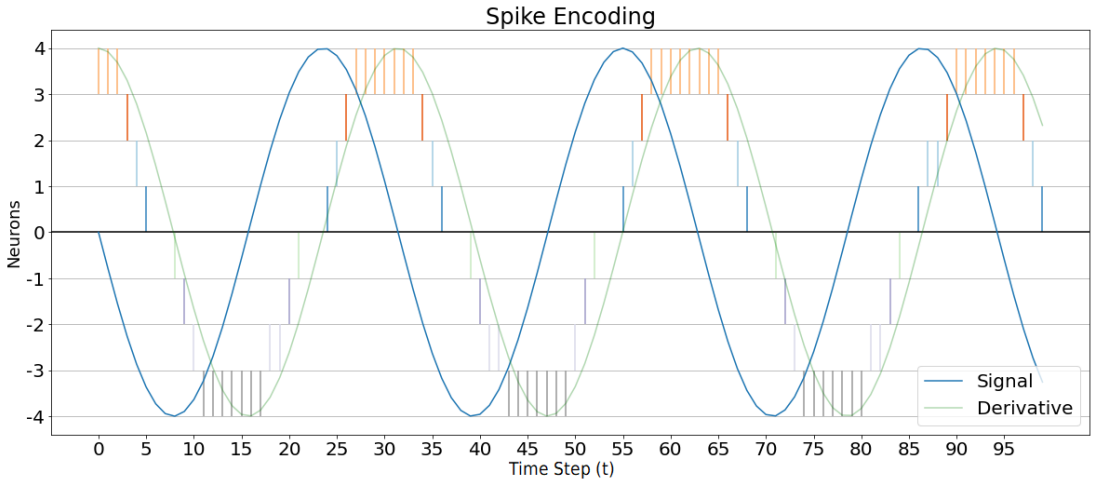


Fig. 5.6: Example of a sinusoidal input signal with its derivative and spike-encoding. The sin (blue) and cos derivative (light green) have been scaled to match the height of the plot. Note the concurrent presence of high-grade spikes (upper and lower rows) with higher values in the cosine and the presence of lower-grade spikes when the derivative approaches zero.

approximates the instantaneous rate of change, or the derivative, of $x(t)$. In practical terms, the goodness of such approximation is constrained by the choice of thresholds for the neurons, which directly affects the granularity of the encoded changes, and by the time resolution (sampling frequency) of the datasets. However, considering a single time step interval and a small enough threshold, such an approximation can be relatively accurate. Fig. 5.6 intuitively shows the goodness of the approximation when the input signal is a sine wave (hence with a cosine derivative). Here, the amount of change in the original sinusoidal signal is encoded by different neurons in the population. It can be seen how the activity from different neurons closely follows the cosine curve (light green).

By means of this encoding system, there are three advantages. Firstly, by taking the (approximate) derivative of the signal, its rate of change is being considered. This tells one how fast the signal will increase or decrease, regardless of its absolute value at a certain point in time. Secondly, by means of the encoding, a differencing transform is practically applied to the input signal, which, in the context of time series analysis, helps increase stationarity and thus make the forecasting of the signal more precise and reliable [275, 276]. Furthermore, envisioning the encoding system as the model for a sensing system deployed to collect time series data, this operation which is normally performed as a preprocessing step, happens for free as part of the sensing. Finally, the amount of information that is propagated through the network is potentially reduced thanks to the presence of a thresholding system that leaves out small changes in the time

series.

5.2.2 Learning and Loss Functions

For the learning part of this work, a simple two-layer fully connected neural network architecture is developed using Intel's LAVA framework [183]. More specifically, the SNN consists of two layers of fully connected LIF [70] neurons, which can be represented by the discrete system in Eq. (2.20). The neuron is also paired with a reset mechanism that resets the voltage to zero whenever a spike is emitted. While in Chapter 4 it has been shown that better-performing spiking neuron model alternatives might exist, the LIF neuron is arguably the most widely used in the literature, hence making it a good choice for future benchmarking purposes. Other than this, it can represent a valid option to increase efficiency, due to its simplicity of implementation.

Regarding the learning rule, an advanced version of the SLAYER rule is employed to train the SNN in a supervised manner. SLAYER is a widely-used spike-based back-propagation learning rule that allows using surrogate gradients of the neuronal spiking functions to learn weights and delays in the SNN. Furthermore, using SLAYER in the context of LAVA also allows seamless future implementations on neuromorphic chips such as Loihi 2 [76]. The standard way of utilising SLAYER is with the loss function defined by equations (6) and (7) in [24], named SpikeTime in the LAVA framework. Through this, the trains of target spikes and output spikes are convolved in the time dimension with a Finite Impulse Response (FIR) exponential kernel and then compared using MSE. By convolving with the FIR kernel, the loss aims to aid the resolution of the credit assignment problem through time. Further to this, more experiments are carried out utilizing two novel loss functions designed specifically for this time series forecasting task. The first one draws inspiration from the concept of minimizing the differences in the inter-spike intervals (ISI) in the target and output spike train; the second one leverages the information from the encoding paradigm to decode the signal and compare the reconstructed versions of the target and output.

5.2.2.1 ISILoss Function

As outlined above, the ISILoss function is inspired by the concept of minimizing the differences in inter-spike intervals between two spike trains. Ideally, the ISIs should be computed for each spike train and then compared to each other. However, this is non-trivial in a back-propagation environment for several reasons. Two spike trains may have different numbers of spikes and, hence, different numbers of ISIs to compare. A possible solution for this is to adopt placeholders with pre-assigned values where the number of spikes differs. Still, it can be time-consuming and sensitive to the chosen pre-assigned value for the interval. Furthermore, this will likely include non-derivable operations, which could break the gradient calculation chain and result in unwanted behaviours. For this reason, a heuristic method based on transforming the spike trains by means of derivable operations only is adopted. Specifically, a time-step vector R can be defined such that:

$$R = [1, 2, \ldots, N] \tag{5.4}$$

where N is the number of time steps in the spike trains. R is used to re-scale each spike in the spike train by the time step at which it occurred:

$$s_R(t) = s(t) \odot R \tag{5.5}$$

where $\odot$ denotes the element-wise multiplication (Hadamard product) between the spike train $s(t)$ and R. Finally, the re-scaled spike train is cumulatively summed. This is achieved by defining the unitary upper triangular matrix T of size $N \times N$:

$$T = \begin{bmatrix} 1 & \cdots & \cdots & 1 \\ & \ddots & \cdots & \vdots \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix} \tag{5.6}$$

and by performing a matrix multiplication with $s_R(t)$. By doing this, a vector of size N is obtained that contains the cumulative sum of $s_R(t)$. The result is then normalized with respect to R.

$$s_{cs}(t) = (s_R(t) \cdot T) \odot \frac{1}{R}. \tag{5.7}$$

The final loss is calculated as a mean squared error of the resulting vectors:

$$L(s_{cs}(t), s^*_{cs}(t)) = \frac{1}{N}\sum_{i=0}^{N}(s^{(i)}_{cs}(t) - s^{*(i)}_{cs}(t))^2 \tag{5.8}$$

where s^*_{cs} denotes the target spike train, transformed as discussed. The idea behind such a heuristic is that by utilizing a cumulative sum, all the time steps present in the spike train contribute to the final loss calculation by some value (likely) different from zero. This allows carrying along some information about the cumulative time of the last spikes with each time step, hence assigning some weight to their role in the spike train, even if no spike was present.

5.2.2.2 DecodingLoss Function

The DecodingLoss is a function that builds on top of another piece of knowledge from the time series encoding: the meaning of each neuron's firing. As a matter of fact, by means of the derivative encoding, each neuron will be assigned a specific threshold, and will encode values that fall in the range $V^{(i)}_{th} \leq x(t) < V^{(i+1)}_{th}$. This piece of information can be used to reconstruct a signal starting from some output spike train $s(t)$ (see Section 5.2), and compare it with the decoded target spike train $s^*(t)$. This is possible by following a similar paradigm as in Section 5.2.2.1. Firstly, it is necessary to define a vector of values that correspond to each neuron's threshold:

$$V = \left[-V^{(M/2)}_{th}, \quad \ldots, \quad -V^{(1)}_{th}, \quad V^{(1)}_{th}, \quad \ldots, \quad V^{(M/2)}_{th}\right]^T \tag{5.9}$$

where $V^{(i)}_{th}$ denotes the (positive) threshold of neuron i. A convenience unitary vector A of size M that can be used to perform a neuron-wise addition per each time step is also defined:

$$A = \left[1, \quad \ldots, \quad 1\right]^T. \tag{5.10}$$

In the experiments, tests are performed both with and without this step in the DecodingLoss function. Finally, the previously defined unary upper triangular matrix T is utilized to perform the cumulative sum. The final reconstructed output is thus obtained as:

$$s_{rec}(t) = (A \cdot (s(t) \odot V)) \cdot T \tag{5.11}$$

where $s_{rec}(t)$ is the reconstructed output and can be either of size N or $M \times N$ depending on whether the matrix multiplication by A was performed. Finally,

the MSE is computed on the reconstructed output and target output:

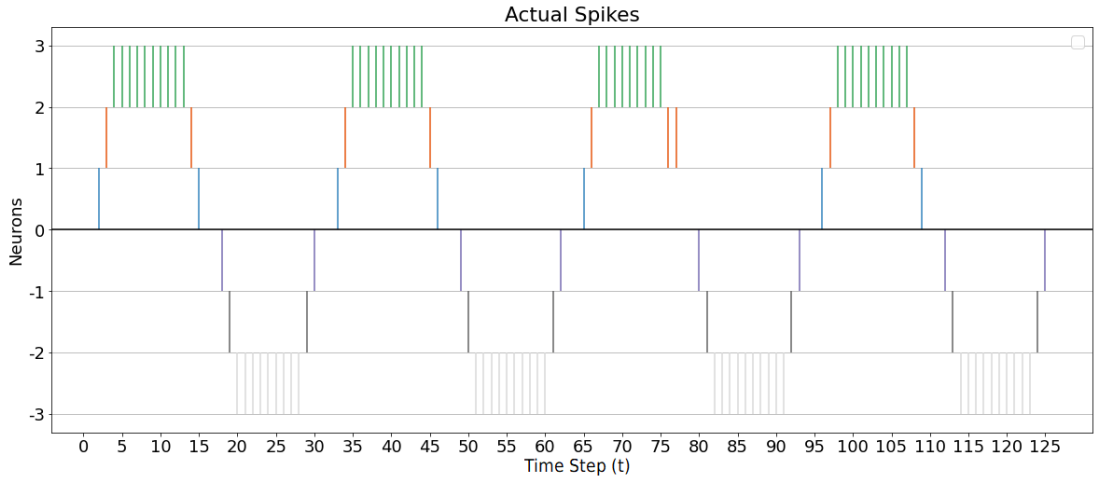
$$L(s_{rec}(t), s^*_{rec}(t)) = \frac{1}{N}\sum_{i=0}^{N}(s^{(i)}_{rec}(t) - s^{*(i)}_{rec}(t))^2 \tag{5.12}$$

where $s^*_{rec}(t)$ denotes the reconstructed target signal. In this case, the cumulative sum has a more physical meaning than the ISILoss, as each value thus obtained directly corresponds to the value of the reconstructed signal due to all the previous changes. It is thus possible to make a comparison of the two signals straight in the signal's original domain and calculate the MSE loss on the final result of the obtained spikes. This theoretically opens up to the SNN learning spike trains that are not precisely the same as the target spikes, as long as the final result is still correct.

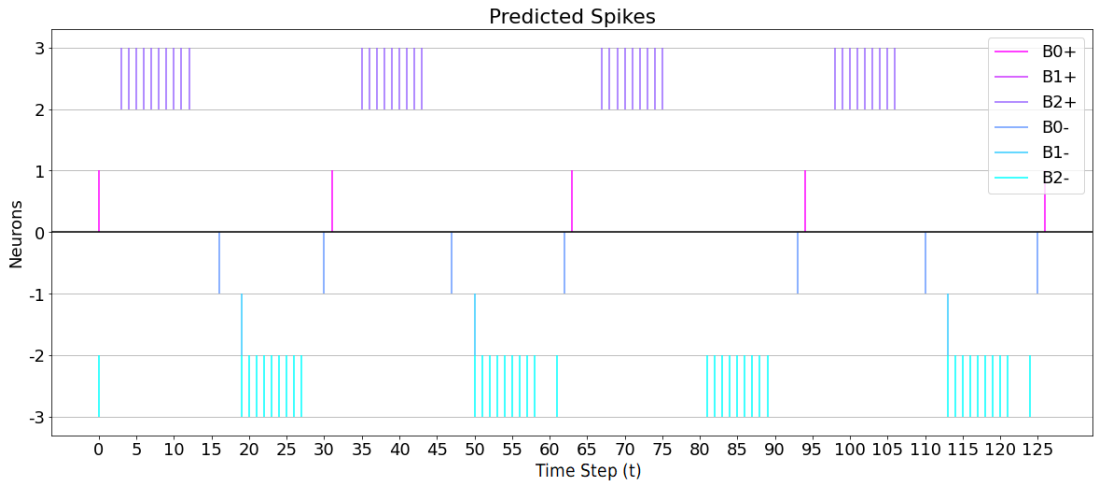
5.3 Results

In order to evaluate the ability of the designed system to learn the prediction task, a series of experiments comprising a number of different settings that span across different values of parameters is devised. In the proposed pipeline, the data is first split into smaller segments (or windows) before being encoded. Experiments are carried out using 128 and 256 time steps-long segments, and each segment has an overlapping of 75% with the preceding one. Using this type of windowing operation helps augment the overall dataset, and helps the network build a more complete representation of the data. For each input segment, a target one is generated by looking at one time step ahead, effectively creating a challenge for the network to learn the upcoming spike. A first batch of experiments is carried out utilising a generated sinusoidal signal as an input, similar to what is shown in Fig. 5.6, as a means to determine whether the system, as devised, would be able to learn the task. A qualitative result is reported in Fig. 5.7 and 5.8, where a target spike train is compared with the prediction from the SNN and their respective decoded versions.

In the figures, the predicted spike train follows the periodicity found in the target train (Fig. 5.7), aside from some mistakes, and the overall reconstruction closely matches the target one (Fig. 5.8), thus showing how the SNN has learnt to predict the next change in the signal. It is therefore possible to proceed to cross-experiment the SNN with different combinations of segment lengths, number of encoding neurons and loss functions. Each combination of parameters is repeated 150 times, for a total number of experiments exceeding 15,000. To evaluate the



(a)



(b)

Fig. 5.7: Actual (target) spikes (5.7a) vs. predicted spikes (5.7b). In the legend, B<N><+ or -> denote the significance of the spike bursts of each neuron. Each row on the y-axis represents the output of a different neuron in the encoding layer. As the number increases, the spike represents larger multiples of the initial threshold set in the encoding. The sign denotes whether the represented change is positive or negative.

quality of the prediction, the decoded output is compared with the decoded target sequence.

To better understand the interplay of the number of neurons with the choice of loss function on the overall performance of the SNN, here is a report and analysis of the boxplots of the MSE of the reconstruction of the output signal with respect to the reconstructed target signal grouped by the number of encoding (and thus decoding) neurons, for the case with 128-long segments. As Fig. 5.9 and 5.10 demonstrate, while the ISILoss does not seem to enable achieving better performances than the SpikeTime Loss, possibly due to an oversimplification

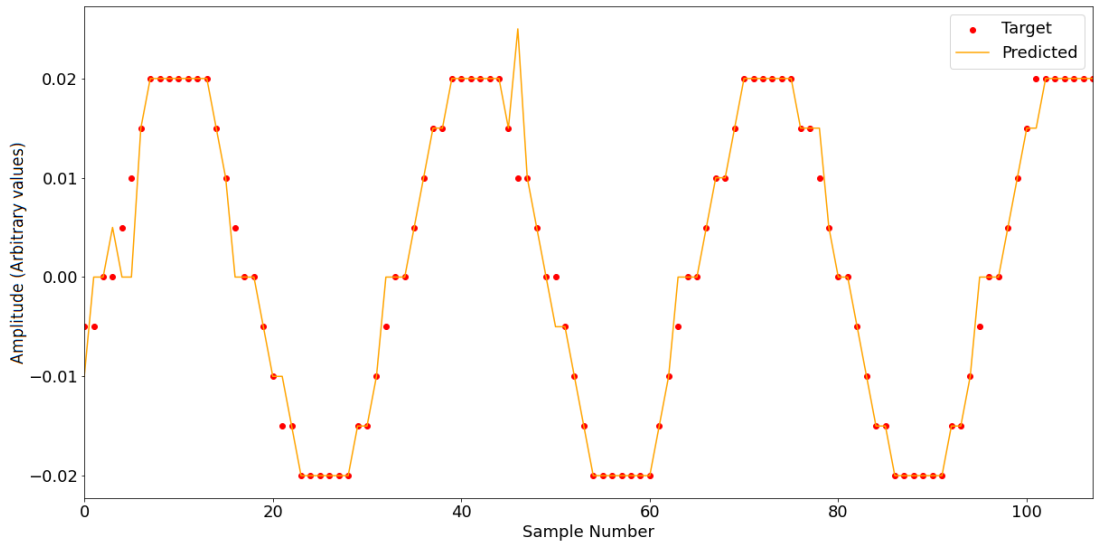


Fig. 5.8: Decoded predicted signal with targets overlaid. Note the resemblance between the two (aside from a few mistakes) and the correct prediction of periodicity.

of the initial concept to suit backpropagation needs, the DecodingLoss steadily attains lower MSE with all hyper-parameter settings, in both datasets. As the number of neurons increases, however, the overall error does too in all cases. The overall performance trend across experiments with different numbers of encoding neurons is reported in Tab. 5.2 and 5.3. Here, it can be appreciated how the Decoding Loss achieves better results and consistently outperforms the other two counterparts. From the tables above, an interesting observation is that the overall errors increase with the number of encoding neurons with all the loss functions. This is not surprising as, given that the task is evaluated against an encoded and then decoded target rather than the original signal itself, the number of neurons that need to be trained to perform prediction is greater. In practical terms, this results in a more complex task, as would be a standard classification task with an increased number of classes. This particular effect is further discussed in Section 5.4.

Motivated by the results above, the DecodingLoss-based solution is selected for further comparisons with other more classical methods like SARIMA (see Chapter 3). SARIMA is a model that is able to account for trends and seasonality in the data, effectively making it a suitable option amongst the classical methods for the data employed in this study. Through this comparison, the objective is to provide a more solid context as to what the performance of the proposed solution signifies and to provide a useful benchmarking point for future reference. In order to use SARIMA, a prior must be known regarding the periodicity of the data, which is inputted into the model prior to fitting. Furthermore, in order to ensure

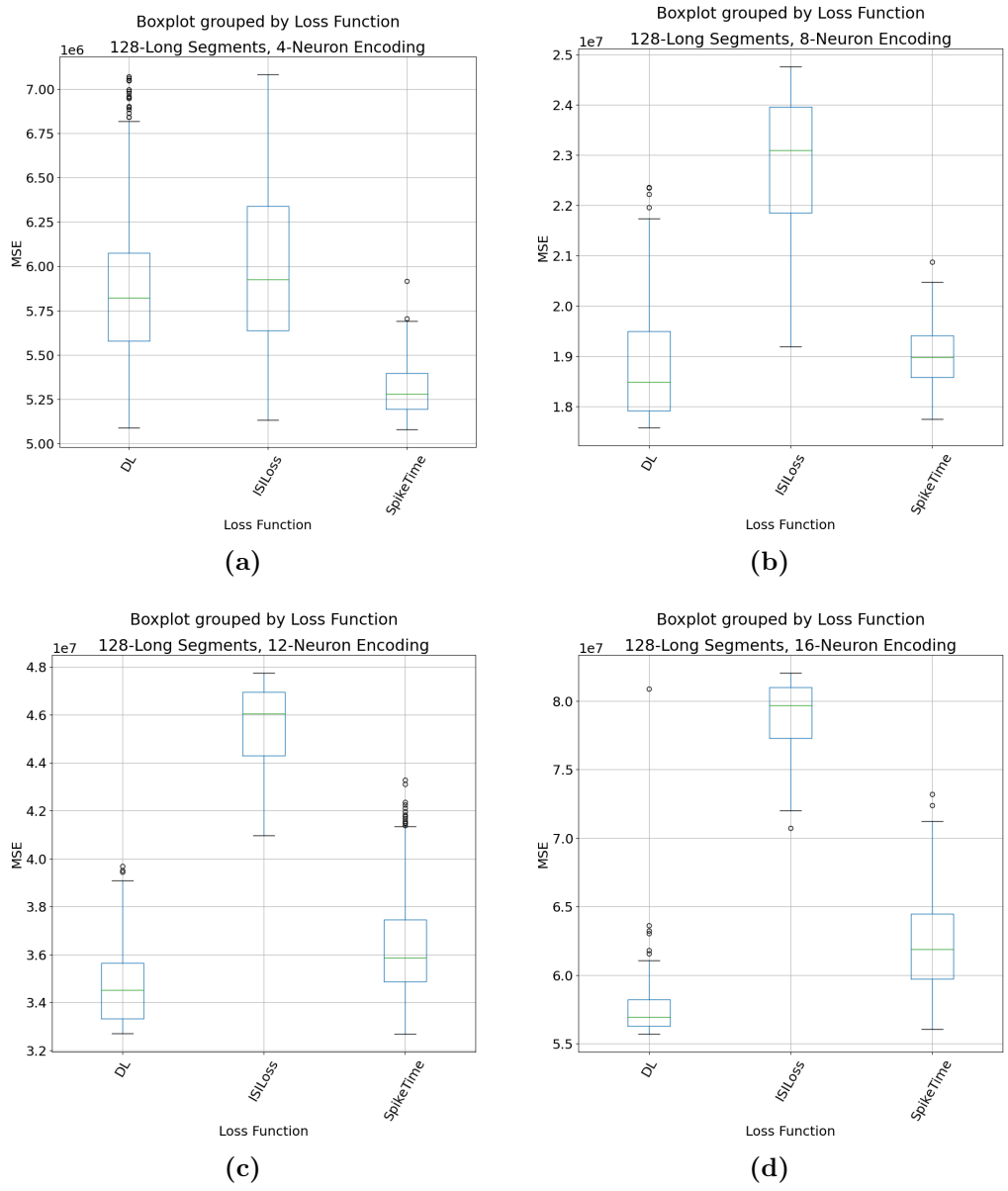


Fig. 5.9: Boxplot comparison of MSE on the Panama dataset for every loss function, grouped by segment length and number of encoding neurons.Note how the DL (DecodingLoss) and the SpikeTime loss both seem to achieve lower levels of MSE, but the DL does so more steadily across different runs (with the exception of the 4-neurons encoding in Fig. 5.9a).

a fair comparison, a hyperparameter search is performed to find the best-fitting parameters. Experiments are carried out in conditions similar to the experiments above, with the task set at predicting the next time step in the series. To obtain single-time step predictions, the model is refitted on the newly observed data every time a prediction is made. Results on the Panama dataset show that the best SARIMA found via the parameter search step achieves an average MSE

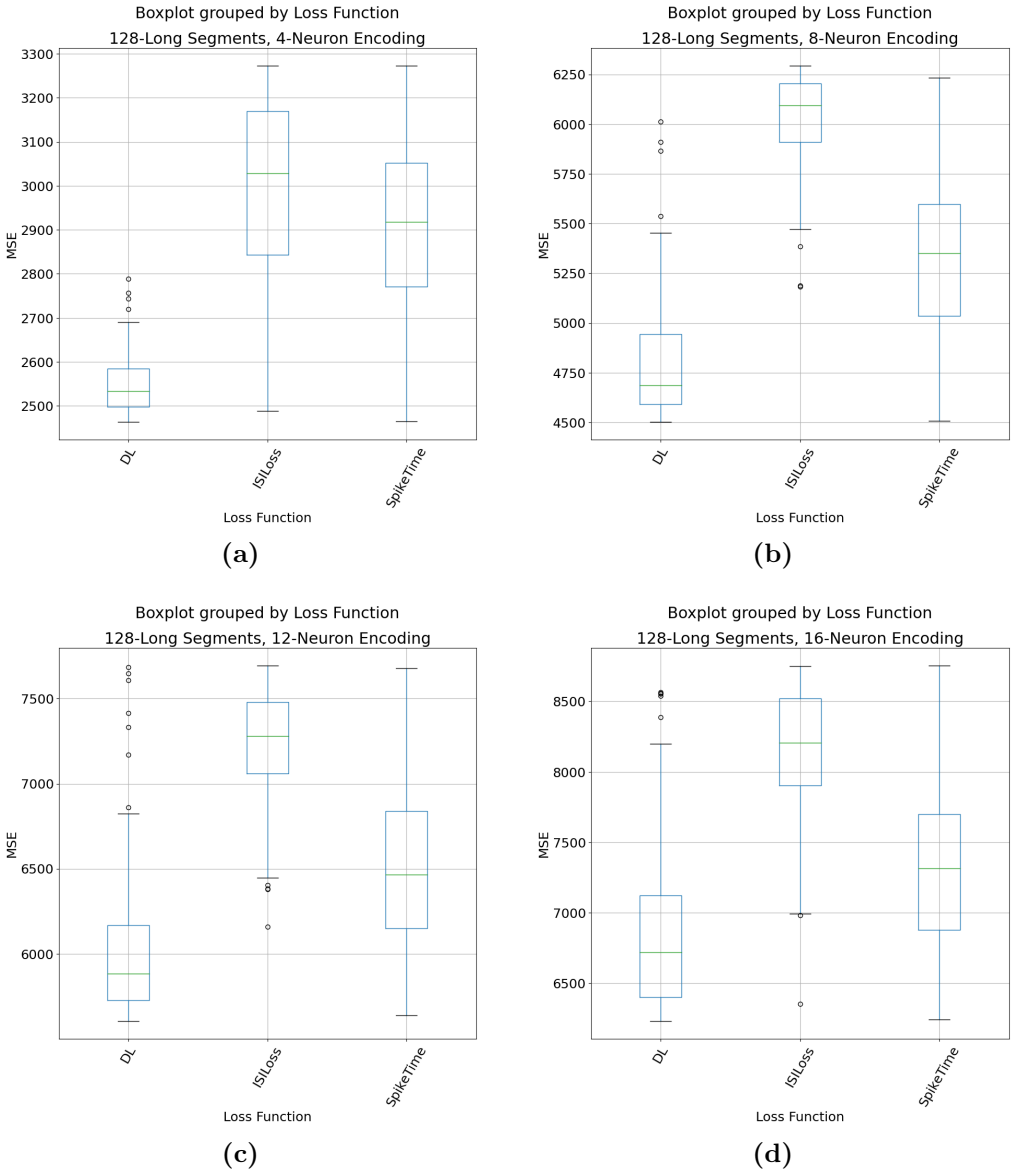


Fig. 5.10: Boxplot comparison of MSE on the ETTh1 dataset for every loss function, grouped by segment length and number of encoding neurons. The trend is similar to the Panama dataset, highlighting a consistent pattern.

between predicted and actual points of 4948.31, whereas the proposed method reaches an average of 2531.03. An example of a prediction relative to these results is given in Fig. 5.11. On the ETTh1 dataset, the SARIMA model achieves an MSE of 1.72, whereas the proposed model sits at 0.39. An example prediction of this is reported in Fig. 5.12.

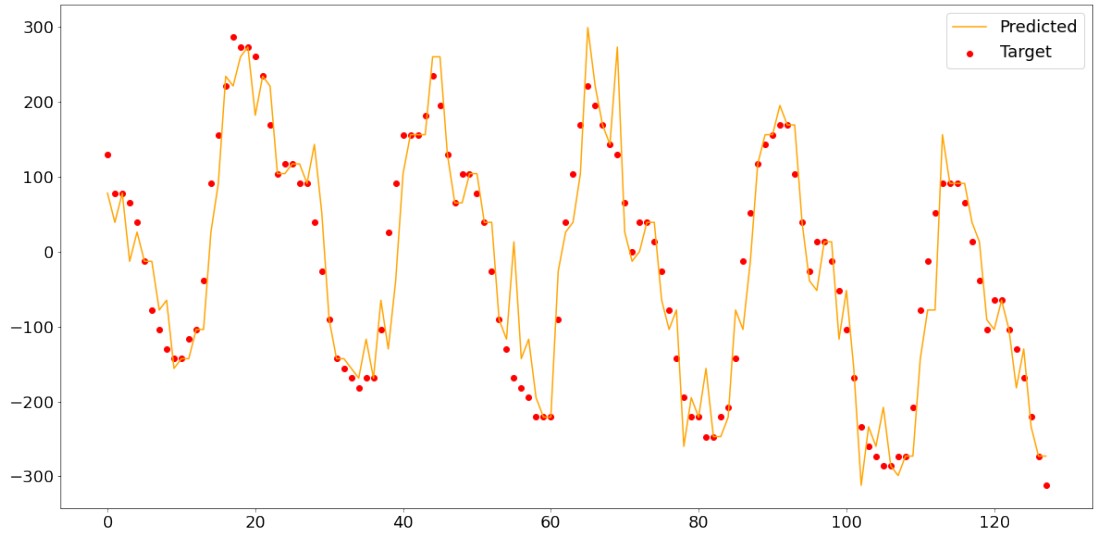


Fig. 5.11: Example from the Panama dataset of values predicted by the proposed solution trained with the DecodingLoss function. The red dots represent the target data points, whereas the orange line represents the reconstructed prediction.

Tab. 5.2. Collated results on the Panama dataset with each loss function against the number of encoding neurons. In each cell, the average reconstruction MSE (rounded to the closest integer) is reported, with the best results highlighted in bold for each different number of encoding neurons.

Loss Fn/N. Neurons	4	8	12	16
DecodingLoss	302	**938**	**1719**	**2926**
ISILoss	326	1350	2669	4718
SpikeTime	**283**	1033	1988	3252

Tab. 5.3. Collated results on the ETTh1 dataset with each loss function against the number of encoding neurons. In each cell, the average reconstruction MSE is reported, with the best results highlighted in bold for each different number of encoding neurons. The reported values are to be multiplied by 10^{-3}.

Loss Fn/N. Neurons	4	8	12	16
DecodingLoss	**2.52**	**4.51**	**5.77**	**6.55**
ISILoss	3.36	6.71	8.16	8.72
SpikeTime	3.04	5.54	6.65	7.49

5.4 Discussions

Section 5.2 describes the nuances of the encoding system and Section 5.3 highlights the importance of an accurate selection of encoding parameters. What appears clear, is that each dataset or use case requires an ad hoc optimization of such parameters, depending on the key elements that require capturing and on the objective of the task to learn. From a broader perspective, it is possible to

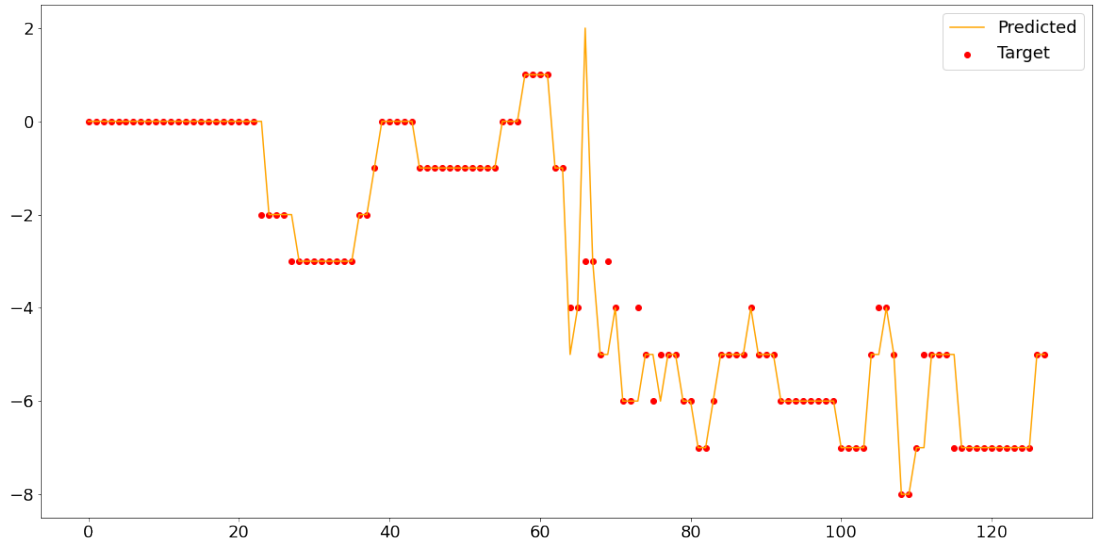


Fig. 5.12: Example from the ETTh1 dataset of values predicted by the proposed solution trained with the DecodingLoss function. The red dots represent the target data points, whereas the orange line represents the reconstructed prediction.

identify two main limit cases. The first one concerns using two neurons only for the encoding step (one for positive and one for negative changes). Here, much of the original information could be lost due to larger changes being encoded in the same way as smaller ones. The second one is about using an infinite number of neurons with an infinitesimal threshold. By doing so, the original signal could be perfectly reconstructed. The first case would be conceivable in a scenario where the task of interest would be to merely predict whether a change is going to take place in either direction, like a simple trend prediction system. The second case (albeit requiring infinite computational resources) could be of interest in high-fidelity settings, like medical imaging, or fine-grain control systems. These two cases represent the two extremes of least computational power and least power of representation and vice-versa, but in most cases, they would not arguably be acceptable and a trade-off between computational power and representational ability must be sought. By assuming another perspective, however, such a trade-off could still be beneficial to other aspects. Let us assume, for instance, to be aware of noise and outliers being present in the data. A typical example would be an underlying current inducing small changes to be read from a sensor, or sudden surges of voltage that are not informative about the phenomenon to measure. In this case, limiting the largest represented value can be beneficial to the prediction system, as well as setting a minimum threshold that would nullify smaller changes. The encoding layer would therefore ignore values outwith the range of interest thus acting as a passive filter for noise, which could otherwise potentially

render the overall prediction process more difficult.

Another point worth discussing concerns the overall decrease in performance as the number of encoding/decoding neurons increases. As previously suggested, the reason behind this observation is likely to be attributed to the increase in complexity that derives from adding more neurons to the system. In the experiments that were devised, an input sequence is encoded into a spike representation and is fed to the SNN. The SNN processes said sequence one time-step at a time and outputs one or more spikes that are apt to predict the signal value in the next time step. The predicted value (spike) is thus compared with the target value depending on the choice of the loss function, and the quality of the prediction is evaluated by comparing the decoded versions of the two. The output of the network, therefore, is not directly compared with the same original time series values in every case, but with its encoded and then decoded version from time to time. This effectively equates to posing a more difficult task to the SNN every time the neuron number increases. As discussed above, fewer encoding neurons would naturally incur a higher quantization error due to their limited representational power, therefore by utilizing the same neurons and comparing them against the original time series' signal, the MSE should be expected to be higher. Instead, by comparing them with an encoded-decoded version of it, the quantization effect is disregarded, therefore simplifying the task. At the same time, more neurons should be expected to incur lower errors, as they could potentially create representations that resemble the original signal more closely. However, for the same reason as earlier, the overall error sits higher than the cases with fewer neurons. Analyzing more in-depth the causes of such effects will be the focus of future research on this topic.

A focal point regards the potential effects of utilising the DecodingLoss as opposed to the SpikeTime loss on the output of the SNN. The SpikeTime loss is built to promote learning to match the exact position (in terms of which neurons have to emit it) and the timing of the spikes. Because each spike is reached by building up a neuron's internal membrane potential, errors are assigned through time, and not just at the moment of the emission of the spike (i.e. also the preceding time steps are affected to a certain degree). When it comes to the DecodingLoss, instead, the focus is on learning a sequence of spikes that, once decoded, achieves the same result as some reference value, or signal, over a period of time. This subtle difference makes it such that the SNN does not necessarily need to learn exactly the timing and position of some spikes, so long that the resulting decoded signal is good enough. This leaves more flexibility to the SNN,

which has therefore access to a greater number of minima to reach, as the combinations of spikes that can achieve this are larger. The degree of freedom allowed by the DecodingLoss is, however, modifiable depending on the modelling needs by including a decoding window in the form of a modified triangular matrix. If a closer resemblance to the target spike train as-is is required, the decoding window can be reduced accordingly to achieve this result. Finally, driven by the results obtained with the DecodingLoss function, the proposed model is compared with a classical method, SARIMA, due to its wide use and suitability to the data at hand. The results of this comparison see the proposed method achieving lower levels of MSE, testifying its effectiveness. Such performance is showcased in Fig. 5.11 and 5.12, where the predicted orange line closely follows the target points (red dots). Apart from serving as a useful benchmarking point for references, this demonstrates the effectiveness of SNNs in performing time series forecasting without any prior knowledge of the type of data utilized (no seasonality prior required) and without the need to refit the model every time a new observation is seen in the test set.

5.4.1 Limitations

The work presented in this chapter envisions an end-to-end sensing and prediction system for time series, but it does come with certain limitations. Neuromorphic technologies have seen a surge in interest and proactive research, but remain in the prototyping phase, lacking a general consensus on best practices. The number and types of sensors and neuromorphic chips are limited, hence the necessity for an encoding layer, as not all data types can be directly sensed by an ad-hoc neuromorphic sensor. This requirement for data conversion into spikes or events introduces an overhead that would not exist if sensors with the same sensing capabilities were available. While this might be acceptable in the current stage of research, it adds an extra layer of complexity.

Moreover, the power and time efficiency advantages highlighted in the literature rely on the vision of an implementation of algorithms within a fully end-to-end neuromorphic system. However, designing or finding such a system remains challenging. As a result, algorithms are often evaluated through simulations, and their power and time efficiencies are often, although not always, estimated based on specific criteria rather than measured directly.

Additionally, this work focuses on predicting one time step ahead in the time series to demonstrate that the encoding system can effectively be used to train an SNN on time series data. More conventional applications typically require

predictions of multiple time steps ahead, which can be a possible direction for future research on this topic.

5.5 Conclusions

This chapter presented an approach to the problem of time series forecasting using SNNs. This was done through the introduction of a novel encoding scheme, inspired by the NM vision sensors, that seamlessly incorporates a differencing transform thus not only encoding information into a spike format, but also performing a free pre-processing step to help make time series more stationary. The encoding system is also shown to be affine with the concept of the derivative of a signal by means of a purposely built sinusoid series. The neuron encoding, in fact, closely resembles a cosine signal. To demonstrate an SNN's ability to learn to predict the next upcoming spike in a so-encoded series, a relatively simple one is built and trained using the SLAYER learning rule. Further to this, two loss functions to be paired with SLAYER and the encoding system are developed, the ISILoss and the DecodingLoss. An initial exploratory analysis of the encoding system is carried out to understand its nuances, and the overall solution is tested on a number of different parameter configurations, so as to obtain more insights on the interplay amongst them. From the obtained results, it is possible to deduce that the encoding scheme can effectively be employed for time series data to be learnt by the SNN, and that different levels of granularity in the encoding resolution can yield different results. In particular, the relationship between the number of neurons in the encoding population and the thresholds assigned to them is discussed. It is also observed that the ISILoss does not perform at the same level as the other two tested loss functions, potentially due to the approximation of the original concept into a version that is much simpler and more amenable to backpropagation. At the same time, the results show a higher consistency in obtaining better prediction results when employing the DecodingLoss rather than the SpikeTime when used in combination with the encoding system. Due to the nature of the DecodingLoss, this suggests that it might be beneficial not to force the SNN into learning a predefined set of spikes to represent some value, but that it could instead prove better to let it find appropriate ways to do so from the point of view of the reconstruction accuracy. To conclude, this chapter proposed a novel NM approach to the forecasting of time series. The encoding leverages ideas from NM cameras, concepts from derivatives of signals, and advantages from the differentiation process, which helps in rendering the

data more stationary. The obtained results demonstrate an ability to learn future spikes from the input data, however, this represents just a first step towards further developments in this direction. Future research will focus on optimizing the loss functions and the encoding system to better learn and represent the data according to the task at hand, and on integrating readings from a number of sources to implement multivariate time series forecasting.

Chapter 6

Resonate-and-Fire Encoding and Classification of Electromyography Signals with SNNs

6.1 Introduction

The accurate and timely classification of EMG signals is crucial for the development of advanced prosthetic devices, rehabilitation tools, and human-computer interaction systems [254]. These systems rely on precise interpretation of muscle activity to translate user intentions into actions, making the accuracy of EMG signal classification paramount [277, 278]. Traditional approaches to EMG signal classification have primarily relied on conventional signal processing and machine learning techniques. These methods typically involve feature extraction followed by classification algorithms such as support vector machines, neural networks, and decision trees. Amongst the traditional signal processing techniques for EMG feature extraction, the FT and WT are often used to extract frequency-domain features. Some examples are represented by the works in [247, 279], where features are extracted and selected in this way before being processed by a classifier. A similar approach is presented in [252], where the authors use an attention-based system on top of the initial FT preprocessing. Although these approaches can be promising in terms of performance, they require preprocessing of the data and subsequent use of computationally hungry solutions like CNNs, which can hamper their applicability in edge, wearable and prosthetic devices. In this regard, Neuromorphic computing could represent a viable option, both in terms of

algorithmic solutions to learn from such complex time-frequency features, and of power efficiency thanks to the underlying NM technology. Previous works have highlighted interest in such applications, demonstrating promising performance, low computational power requirements, and low latency. In [254] the authors build two relatively simple SNN architectures and implement them on the Loihi chip [182]. Not only do they demonstrate the ability of their SNN to learn the task with relatively good accuracy, but they also showcase very low latency and power consumption when actually implemented on the chip. All of the above without the need for any preprocessing of the signal. Another example is given by [255], where the authors use a slightly deeper SNN implemented on a board. Their solution allows them to reach even higher accuracy levels, at the cost of requiring slightly longer data samples.

Chapter 4 proposed a study that highlighted the possibility that utilizing certain types of spiking neuron models in certain contexts could enhance the learning capabilities of an SNN, or NM system more in general. With this consideration, this chapter proposes a novel NM pipeline for the real-time classification of EMG gesture signals. The proposed pipeline relies on the use of Resonate-and-Fire spiking neurons, a special type of neuron which is modelled by a complex internal state $z = a + ib$ and whose dynamics have been demonstrated to approximate the behaviour of a STFT [76] (see Section 2.2.1.2.4). This capability allows for a more nuanced encoding of the dynamic and non-stationary nature of EMG signals, which are typically characterized by their complex temporal patterns and frequency content. By leveraging RF neurons, the encoded signals retain critical information necessary for accurate gesture recognition. To reduce the impact of hand-tuning several hyper-parameters, an optimization step is performed to find heuristically optimal parameters to encode data. To this extent, a decoding mechanism is also introduced that allows the encoded data to be transformed back into its original domain, thus allowing a one-to-one comparison for optimization purposes. The encoding layer is followed by a relatively simple SNN developed in Intel's LAVA [280] framework and trained using SLAYER [24]. This not only facilitates the implementation but also allows for seamless future deployment onto hardware platforms thanks to the features exposed by the framework. The proposed pipeline is tested using the Ninapro DB5 dataset [25] which a comprehensive repository of EMG signals recorded during various hand movements, providing a rich source of data for training and validating classification models. This dataset includes recordings from multiple sensors placed on the forearm, capturing a diverse range of gestures performed by different subjects. Such a

dataset is ideal for exploring the encoding abilities of the RF neurons and the overall performance of the proposed system due to its challenging nature. In order to explore the interplay of the different components and parameters, a number of experiments were carried out on different subsets of the dataset. The obtained results reveal that the proposed method not only achieves accuracy levels higher than the other SNN-based approaches, but also higher than the conventional DL-based ones on the same dataset. To summarize, this chapter presents the following contributions:

- A novel EMG classification framework that achieves state-of-the-art performance levels thanks to the use of RF spiking neurons and SNN.
- A decoding system from the spike-frequency domain back into the temporal domain that can enable direct comparison for optimization and interpretability purposes.
- An exploration of the impact of some of the experimental parameters that highlight interesting insights in the presented pipeline.

The rest of the chapter unfolds as follows. Section 6.2 presents the steps taken to prepare the data; Section 6.3 details the nuances of the spike-frequency encoding and decoding process; Section 6.4 provides a description of the employed SNN model; Section 6.5 details the performed experiments and obtained results; Section 6.6 summarises the work, highlights the most important findings, and outlines possible practical implications and future works.

6.2 Data Preparation

For the purpose of this work, the Ninapro [25] DB5 dataset, a commonly utilized dataset in the realm of Electromyography gesture recognition, was utilized as a benchmark. In this dataset, EMG activity was recorded using two Thalmic Myo armbands [259], each equipped with eight single differential electrodes for surface EMG (sEMG) and a 9-axis inertial measurement unit (IMU). The armbands sample data from the eight sEMG sensors at a frequency of 200 Hz with an 8-bit signed resolution, streaming the data to a computer via a Bluetooth low-energy connection using the Myo Connection application.

In the experimental setup, the subject wore the two Myo armbands placed adjacently. The upper Myo armband was positioned near the elbow, with the first electrode on the radio-humeral joint, in accordance with the Ninapro electrode

configuration [25]. The lower Myo armband was placed just below the first, closer to the hand, and tilted by 22.5 deg to fill the gaps left by the electrodes of the upper armband.

The dataset comprises 53 different classes (52 + rest position) subdivided into three exercises (A, B, and C) depending on the type of action participants were asked to perform. Each action was repeated a total of 6 times by each participant, and every repetition was interspersed by the resting position. Because participants were required to go back to the rest position every time, this created a strong imbalance in the dataset where $\sim 60\%$ of the samples are resting positions, and the remaining percentage is subdivided amongst the remaining 52 classes, which have a relatively low number of samples with respect to the number of gestures. Furthermore, the spectral decompositions for some of the performed gestures are nearly identical [281]. For these reasons, the Ninapro DB5 dataset is often considered a challenging EMG dataset.

The first step for the experimental procedure is to subdivide the data samples into 16-channels, 260 ms-long chunks with an overlapping of 80%, so as to increase the total number of samples available for training. This value was determined in accordance with [256] to achieve real-time classification responses. Because the signals are sampled at 200Hz, this results in chunks with 52 data points each (time steps). Furthermore, as a means to augment the dataset, samples are randomly delayed (anticipated) by a value ranging between -8 and 8 time steps. Delaying data samples any further would prove pointless due to the overlapping utilized when creating chunks. No further pre-processing is applied to the data. In the experimental pipeline, repetitions 2 and 5 are reserved for validation, as per Ninapro standards [25], while the rest is employed for training.

6.3 Spike-Frequency Encoding

Electromyography data commonly takes the form of a sequence of real-valued data points collected at a given sampling frequency and with a certain number of channels (sensors) that are carefully positioned to pick up muscular activity [282, 283]. Muscular activity generates electric signals that possess distinct frequency characteristics based on the specific muscle and the movement involved. Consequently, different actions can theoretically be distinguished by analyzing these time-frequency signatures. In order to do this, we conceive a methodology that relies on the use of Resonate-and-Fire spiking neurons. When used with the right parameters, RF neurons have been shown to exhibit properties similar to

the STFT [76], effectively approximating the transform in representing features in the frequency domain. Unlike the STFT, however, the RF neurons do not abide by the rule that a higher frequency resolution results in a lower temporal resolution. Instead, they can achieve high frequency resolution, while maintaining high temporal resolution at the same time. An example of the STFT approximation ability is reported in Fig. 6.1. Here, the STFT of a signal is visually compared with the output spikes of a population of Resonate-and-Fire neurons. We base our implementation of the RF neurons on the functionalities present in the LAVA [280] framework, but with some modifications to the hyperparameter availability and initialization process. In particular, in our implementation, RF neurons can be assigned variable decay rates, as well as having a customized set of periods (resonating frequencies). These two additions helped achieve more useful encoding. From the initial experimentation, in fact, being able to assign different decay rates to neurons with different resonating frequencies allowed for a cleaner (less noisy) frequency encoding in that RF neurons would be more specific for the frequency band they would be firing for. At the same time, the possibility to assign a custom set of periods enables focusing on frequencies that are known to be representative of the data of interest, thereby allowing a bank of RF neurons to act as a passive band-pass filter. As seen in Section 2.2.1.2.4, a Resonate-and-Fire neuron is defined by Eq. (2.24). From the equation, it is notable how the spiking function is set to emit $\Re(z_k[t])$, i.e. the real part of the neuron's internal state, when the conditions are met. This is important as such value can be related to the magnitude of the frequency at that moment in time, similarly to a spectrum. This slightly differs from the more conventional conception of spiking neurons emitting either a 0 or a 1 as a spike; however, recent technological advances [76] have shown that the transmission of values greater than 1 between neurons can easily be implemented in NM chips at a relatively low cost. By defining a bank of neurons with varying resonating frequencies and decay rates, it is possible to obtain an encoding layer that reads an EMG signal in the time dimension and translates it into a sequence of frequency-sensitive spikes emitted by a number of neurons. After an initial tuning phase, this work employed a set of RF neurons, aimed at capturing the most relevant frequency information from the EMG signals. In particular, frequencies are offset at 20 Hz. This is because, below this threshold, wider movements and instrumentation artefacts are known to be the cause of noise. As such, by not including lower frequencies, the bank of neurons effectively acts as a passive band-pass filter. The highest parametrized frequency is 100 Hz. This is to avoid incurring aliasing phenomena due to the dataset being

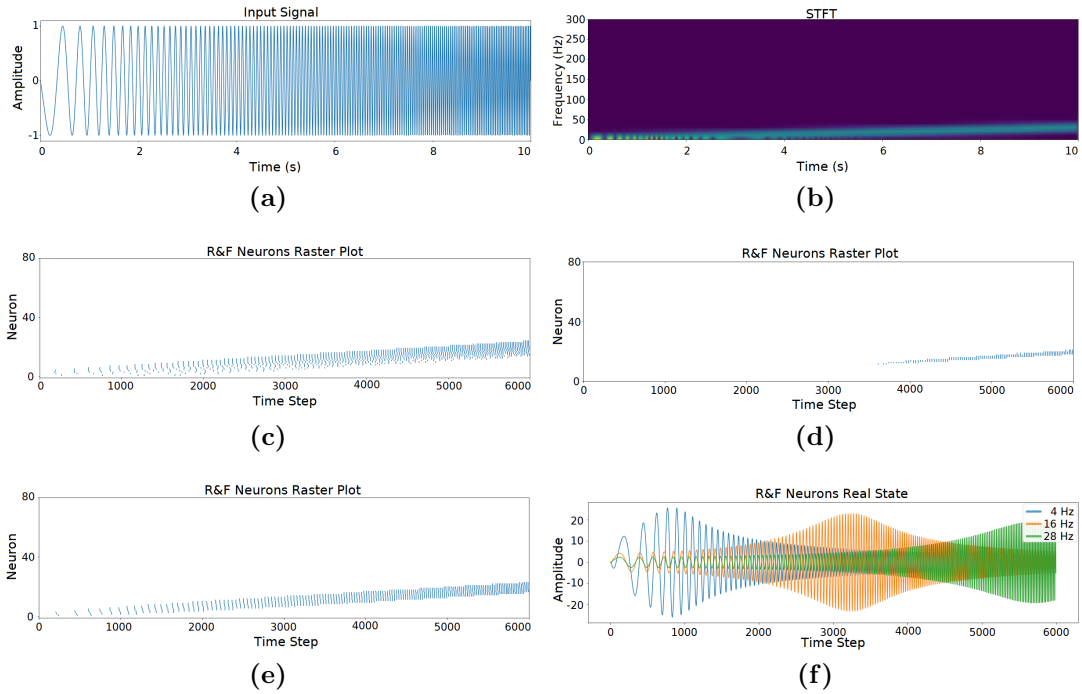


Fig. 6.1: Example of RF spike encoding with different decay values. 6.1a is the input chirp signal with frequency increasing from 1 to 30Hz; 6.1b is the STFT of the chirp; 6.1c is the spiking output using a low decay rate; 6.1d is the spiking output using a high decay rate; 6.1e is the spiking output using a linear decay range; 6.1f is the real part of some of the RF neuron's internal state for the linear decay case. Notice the differences between the outputs with different decays. A too-high decay (6.1d)tends to leave out lower frequencies; a too-low one (6.1c) tends to over-represent them and introduce noise. Finally, notice the internal state of the neurons having a stronger resonance in correspondence with their resonant frequencies.

recorded with a sampling frequency of 200 Hz.

The RF neuron class calculates the resonating frequency of the neurons based on a specified period, which is represented by a number of discrete time steps. To automate the generation of a set of periods for a linear range of frequencies, the following formulation is employed.

Let linspace(a, b, n) denote a function that creates a vector of n equally spaced values in the range $[a, b]$:

$$\text{linspace}(a, b, n) = \left\{ a + \frac{k(b-a)}{n-1} \mid k = 0, 1, 2, \ldots, n-1 \right\}, \tag{6.1}$$

and let f_s and f_o be the sampling frequency and frequency offset respectively. The vector of resonating frequencies is thus defined as

$$\mathbf{f} = \text{linspace}(f_o, \frac{f_s}{2}, N), \tag{6.2}$$

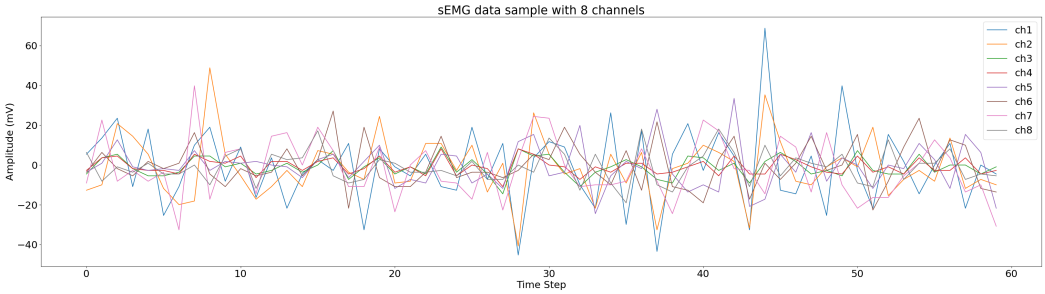
where N is the number of encoding neurons. To obtain the number of time steps required to represent the frequencies in $\mathbf{f}$, it is enough to take its inverse and multiply by the sampling frequency f_s:

$$\mathbf{T} = \frac{1}{\mathbf{f}} \cdot f_s. \tag{6.3}$$

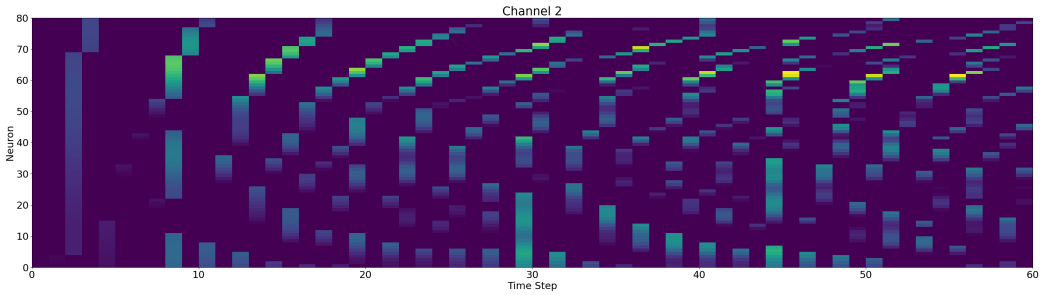
This is a required procedure, because, ultimately, the neuronal dynamics of the RF neuron are based on the smallest unit of time possible, i.e. the time step. The real value of such a unit is unimportant for computations and is dictated by the sampling frequency or by the specifications of the final system. The represented frequency is thus left to interpretation depending on the particular use case. For instance, a period of $T = 4$ time steps in a scenario where the sampling frequency is $f_{s1} = 100\,Hz$ would correspond to a frequency $f_1 = \frac{1}{T} \cdot f_{s1} = 25\,Hz$, whereas in the case of $f_{s2} = 200\,Hz$ the corresponding represented frequency would be $f_2 = 50\,Hz$.

The decay in a RF neuron is also an important factor to consider when designing a bank of encoding neurons. This is because the decay directly affects the sensitivity of some neurons to certain frequencies. As mentioned earlier, the RF neurons are an approximation of the STFT rather than an exact equivalence. One way in which this approximation takes place is in neurons emitting spikes for frequencies close to their resonating one. Together with the choice of threshold, the rate of decay modulates this behaviour. In practical terms, the internal state of a RF neuron can be considered a sinusoidal signal with frequency f. When an incoming signal resonates with the neuron, the amplitude of the neuron's internal state grows. Such growth can also be triggered by nearby frequencies as long as they do not add up destructively. The decay term is what helps mitigate this effect by reducing the value of the internal state at a certain rate, thus restoring it to normal. When the resonating frequency is found, the neuron should be able to have its internal state grow up and beyond the set threshold, despite the action of the decay parameter. It was found that employing a set of different decay rates depending on the resonating frequency helps produce spike trains that are more accurate in terms of their frequency representation, which in turn can result in more easily discernible features thanks to a reduction in noisy (unwanted) spikes. In particular, employing higher decays for higher resonating frequencies appears to be an effective approach. In this work, a linear set of resonating frequencies was utilized, therefore, decay rates were also initialized as a set of linearly spaced values; however, the implementation also allows for a logarithmic scale to accommodate future diverse use cases. An example of the resulting encoding applied

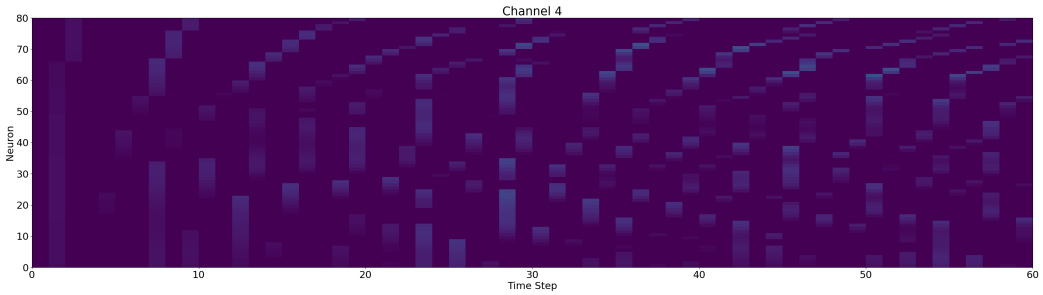
to a sample from the Ninapro Db5 dataset can be found in Fig. 6.2.



(a)



(b)



(c)

Fig. 6.2: Example of RF neuron encoding on a Ninapro DB5 Exercise A sample. Fig. 6.2a reports the sEMG signal across the 8 different channels for one time window. Fig. 6.2b and 6.2c report the encoding corresponding to channels 2 and 4, respectively. Channel 2 appears to have much higher activity within the selected window than Channel 4, and this also shows in the RF encoding, thus likely making Channel 2 a better predictor for this particular sample.

With the aforementioned setup, an encoding layer can be thus defined by a bank of RF neurons with a given set of parameters. The specific values of the parameters were obtained both empirically and by means of a Bayesian optimization round. In order for the optimization to be possible, a decoding paradigm was developed that could transform a spiking output from a bank of RF neurons back into the original signal's domain, so that it could be compared with

the original signal itself to evaluate the encoding quality. The decoding process leverages the resonating frequency and decay rate of each neuron to create a set of convolutional kernels with values given by an exponentially decaying sinusoid according to the following:

$$w_k = cos(2\pi f t) \cdot (1 - \lambda_k)^t, \tag{6.4}$$

where w_k represents the kernel weights for the k-th RF neuron with frequency f, and λ_k is the decay rate for the same neuron. The idea is that if a spike was emitted at a certain point in time, a sinusoid with that neuron's given frequency was then present. However, the sinusoid would not necessarily be always present from that moment on, thus explaining the exponential decay. This transformation can effectively be obtained by means of the correlation operation (enacted by a convolutional layer). The result is a set of N signals with specific frequencies that can then be averaged to obtain the original (reconstructed) signal. To achieve higher precision in the computation, each kernel is required to be as big as the incoming signal, and the signal requires appropriate padding at the beginning and at the end, to accommodate the complete convolution. However, this could potentially become a computational memory burden where the signals are extremely long. Nevertheless, it was empirically found that relatively good quality reconstructions could be achieved by utilizing considerably smaller kernels, as long as the represented decaying sinusoid was allowed to reach values close to zero. For a qualitative demonstration purpose, Fig. 6.3 reports a plot of a single-channel soundtrack (which allows an easier visualization than a multi-channel EMG signal) and the reconstruction from its RF-encoded version by means of the devised decoding paradigm. As anticipated, encoded signals could in this way be compared with their original counterpart by means of a mean squared error metric, thus enabling automated Bayesian optimization. The so-obtained optimal parameters are reported in Tab. 6.1 and have been utilized for subsequent experiments with the Ninapro dataset.

Tab. 6.1. List of optimal parameters for the bank of RF neurons obtained through optimization. The Scale parameter is an internal scaling value utilized in the Lava framework. Where no unit of measure is reported, parameters are considered dimensionless.

Parameter	Value
N. Neurons	80
Threshold	[2, 0.02]
Frequency	[20, 100] Hz
Decay	[0.00774, 0.0933]
Scale	1 <<12

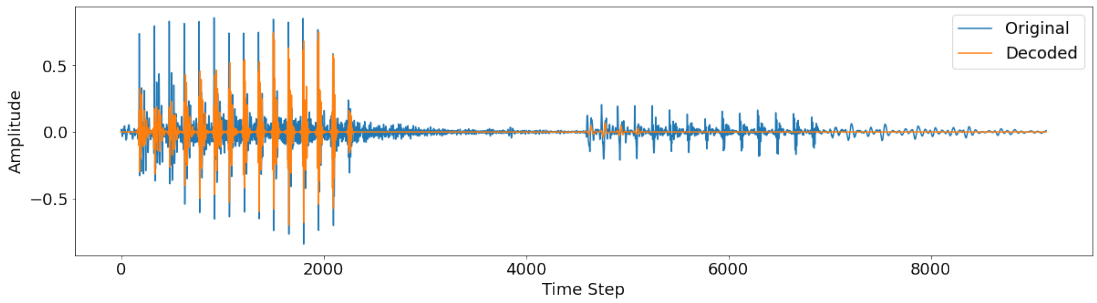
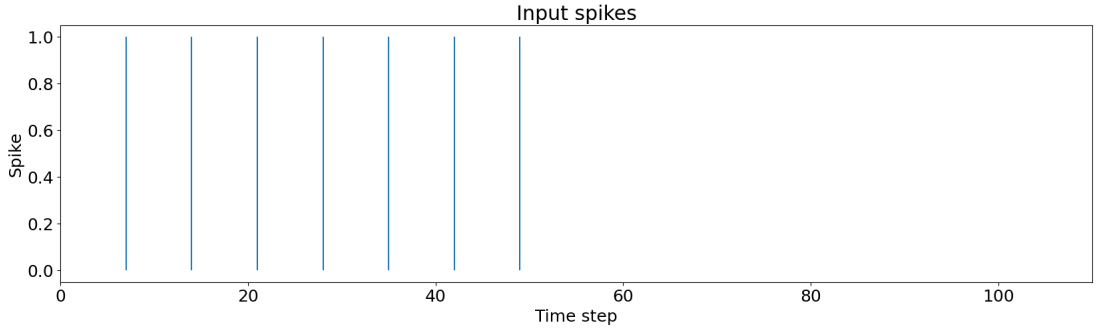


Fig. 6.3: Example of single-channel soundtrack (blue) and its RF encoded-decoded version (orange).

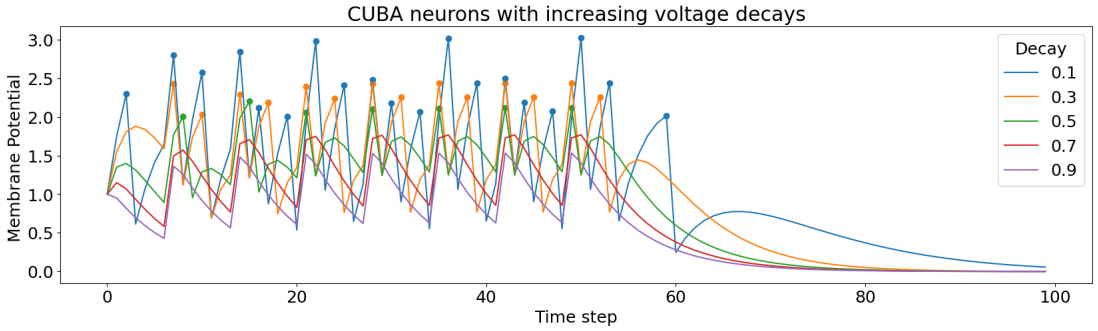
6.4 Spiking Neural Network

Following the encoding layer, the Spiking Neural Network receives the spike-encoded input to process and learn features in the data. The SNN used in this work is a relatively simple 3-layer fully connected architecture that employs CuBa neurons (see Section 2.2.1.2.5). This neuron model is a second-order implementation of the LIF model which accounts for dendritic input decay over time. In practical terms, this is similar to having a decaying memory of the input, which keeps exciting the LIF for a time that depends on its decay rate, thus allowing the creation of potentially more complex feature representations. The dynamics of the CUBA neuron model can be found in Eq. (2.25). In Fig. 6.4 an exploration of different values of α_v with fixed α_u of the CuBa model is reported. Notably, for certain combinations of parameters, the neurons continue spiking even after the input spikes terminate.

The first two layers of the network are designed so that the number of nodes is proportional to the number of ‘channels‘ in the input. In this case, this is dictated by the number N of encoding neurons. The first two layers are also able to learn delays other than just weights by means of the SLAYER [24] learning rule. The



(a)



(b)

Fig. 6.4: Exploration of CUBA neuron responses with varying voltage decays. The current decay is maintained fixed at 0.15. Note especially the presence of spikes (dots) in the output (6.4b) even after the input (6.4a) has terminated. This is made possible by the relatively low current decay, which keeps feeding into the neurons' voltages.

final layer is the classification layer, which thus has a number of output neurons depending on the number of different classes, and does not introduce any delay learning. For all the layers, weights are initialized with the initialization scheme described in [284].

Finally, a class weight balance system was adopted. To be specific, class weights were calculated according to the following. Let C be the total number of classes and n_c with $c \in \{0, 1, 2, \ldots, C-1\}$ be the number of samples in the training set per each class, then the weight for class c is defined as

$$w_c = \gamma \cdot \frac{\sum_{c=0}^{C} n_c}{N \cdot n_c}, \tag{6.5}$$

where $\gamma \in (0, 1]$ is a scaling factor. When evaluating error in the loss function, each class would then be multiplied by a so-obtained weight.

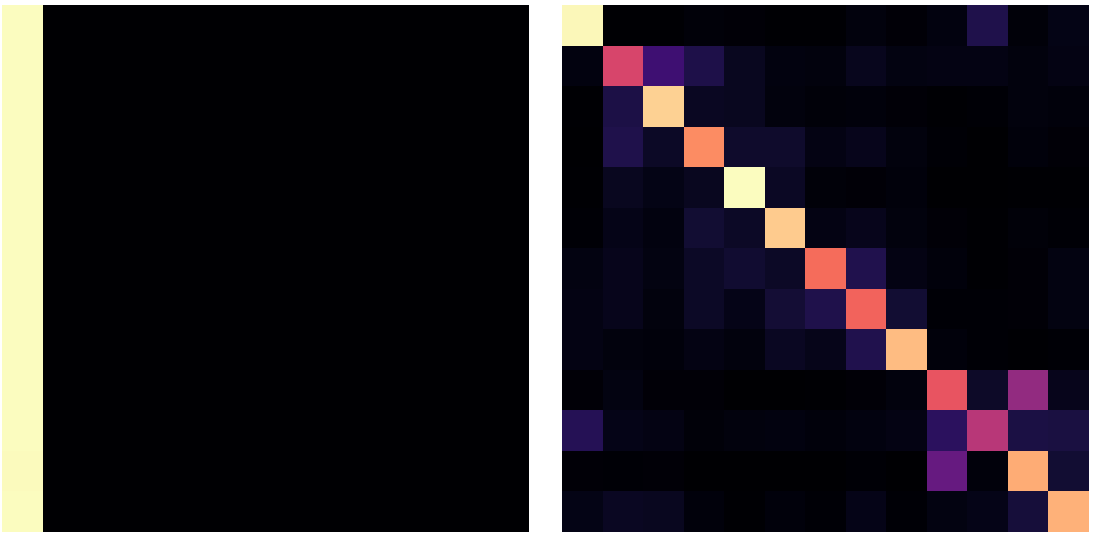


Fig. 6.5: Example of bad (left) vs good (right) prediction results. The heatmap represents a visual representation of the (normalized) confusion matrix obtained by training the SNN on exercise A without any class weight (left) and with class weight (right). In the left-hand figure, numerical results report the SNN as being relatively accurate ($\sim$ 70%), but in reality, it only learns to remain silent and thus classifies everything as "rest" (class 0).

6.5 Experiments and Results

Due to the imbalanced nature of the dataset, developing a training pipeline that would avoid overfitting was a non-trivial task. Early experiments showed how simply presenting the data to the network would lead to the SNN learning to either remain silent (which is equivalent to predicting class 0, i.e. the 'rest' position) or to activate one output neuron only all the time (in case the 'rest' position was assigned a different class index) very quickly (Fig. 6.5). To overcome this issue, the SNN's hyperparameters required extensive fine-tuning. The biggest contribution, however, was given by the use of class weights for the calculation of the classification loss as described earlier. Given this initial setup, experiments were carried out with a number of different configurations and dataset splits. As anticipated in Section 6.2, repetitions 2 and 5 were reserved for validation purposes, whereas the others were utilized for training. All the participants' data was used for training at all times and, as per common practice, the dataset was subdivided into Exercise A only (12+1 classes), B only (17+1 classes), C only (23+1 classes), and all together (52+1 classes). Each training session was allowed up to 500 training epochs to learn. As a way to compare with relevant benchmarks in the literature [252, 254, 255, 281], initial experiments focused mainly on Exercise A, which relates to finger gestures. A specific comparison of

Tab. 6.2. Comparison of the performance on Ninapro DB5, exercise A. Latency considers EMG segment length and reported processing times.

Reference	Model	Latency	Accuracy
Vitale et al. (2022) [254]	3-layer SNN	300+5.7 ms	74%
Scrugli et al. (2024) [255]	4-layer SNN	500+31 ms	85.6%
Chen et al. (2020) [281]	4-layer CNN	260 ms + processing	69.62%
Shen et al. (2022) [252]	Conv. Visual Transformer	200 ms + processing	76.83%
This work	3-layer SNN with RF neurons encoding	260 + 30 ms*	**92.36%**

*Estimated time considering limitations imposed on the system, it does not consider hardware measurements.

the performance in this setting can be found in Tab. 6.2. Notably, the accuracy levels achieved through the methodology employed in this work are consistently higher than the references, whilst maintaining a latency lower than 300 ms, thus remaining in the real-time feasibility domain. This not only applies to other SNN-based works, but also to conventional DL and attention-based ones, which are more commonly found to be better performing in the literature. It is to be noted that the latency reported for this work is based on calculations of propagation times and delays, and is not measured from a hardware implementation, which could result in a different value. Furthermore, the comparison is performed on the solutions as whole pipeline as found in the literature. This includes any pre-processing step, being the RF neuron encoding an integral part of the proposed approach. For completeness, Fig. 6.6a reports the normalized confusion matrix as a heatmap for the aforementioned task. Note that there does not seem to be a tendency to prefer a class 0 to other ones. Instead, class 0 is sometimes even misclassified as other classes: this is a direct effect of using the weight scale system, and can be adjusted by varying the scaling factor γ. It is also possible to notice a tendency to mistake class 9 for class 11, and vice versa. This is likely due to the similarities in the spectral decomposition of these classes in the dataset [254].

Overall, the designed pipeline includes several different components that could influence the performance of the system. In particular, the number of channels in the input signal directly affects the amount of information accessible to the SNN and the number of hidden units it has (due to its design). The number of classes to discriminate amongst also represents an important factor, especially in

Tab. 6.3. Performance comparison with different settings and parameters.

Task	N. Channels	N. RF Neurons	Delay	Accuracy
Ex. A	8	80	True	88.05%
Ex. A	16	80	True	**92.33%**
Ex. A	16	80	False	92.20%
Ex. A	16	20	True	88.69%
Ex. A	16	80	True	92.33%
Ex. A	16	160	True	**92.36%**
Ex. A	16	80	True	92.33%
Ex. B	16	80	True	89.78%
Ex. C	16	80	True	82.85%
Ex. A + B + C	16	80	True	75.43%

the Ninapro DB5 dataset where a high class imbalance is present. Other factors such as the possibility of learning delays within the SNN and the number of encoding RF neurons could also affect its ability to learn features from the data. To understand the effects of the above, further experiments were carried out by varying the mentioned parameters. The results are collected in Tab. 6.3, where they are compared against each other.

From the results, it is possible to notice a few points. Beginning with the number of channels, it is clear that utilizing 16 channels (i.e. the input from both the MYO bands) is beneficial to the performance (92.33% accuracy). However, the loss in accuracy that comes from using only one channel is not disruptive, as it still allows the SNN to achieve 88% accuracy, which sits above several other works in the literature, thus potentially allowing a significant reduction in hardware requirements with a relatively small loss in performance. The number of classes, i.e. the types of exercises presented to the network, instead, does have a heavier impact. The SNN's performance seems to decrease with the number of classes with the overall performance on the whole dataset dropping to 75.43%; however, this could be expected due to the increased complexity. Fig. 6.6 reports the heatmaps of the normalized validation confusion matrices for the different cases. The images highlight that the vast majority of the samples are correctly classified and that some classes are more easily misclassified as other ones, likely due to the similarities in some of the EMG signals.

The number of encoding neurons also shows an interesting trend. Experiments with 20, 80 and 160 RF neurons were carried out. As discussed in Section 6.3, the frequency spectrum that was considered in this work ranges from 20 to 100 Hz, thus allowing for 80 integer frequencies. This number was doubled (160)

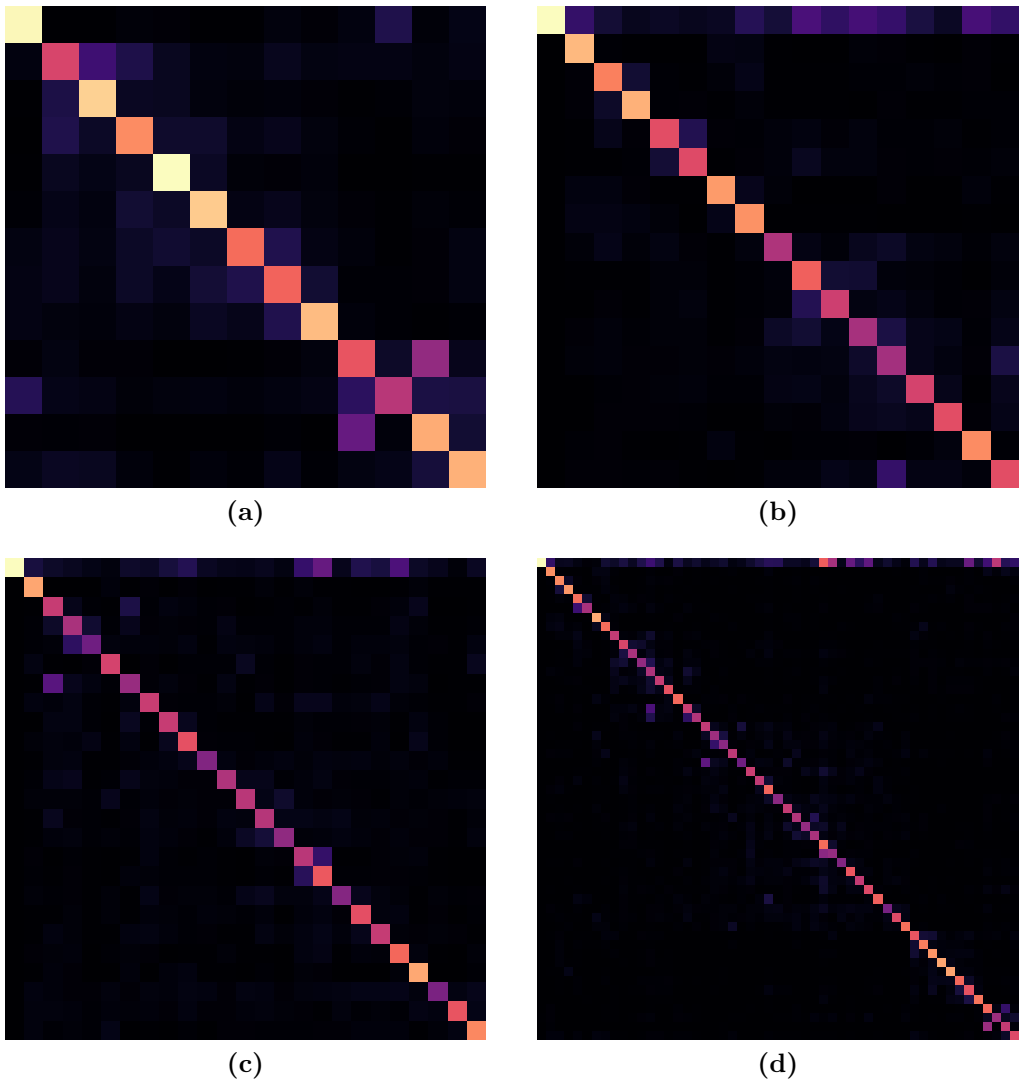


Fig. 6.6: Normalized Confusion Matrix Heatmaps for each of the different tasks. Columns represent predictions, whereas rows are true labels. 6.6a is for Exercise A, 6.6b for Exercise B, 6.6c for exercise C, and 6.6d is for the complete dataset.

and quartered (20) to get an over-representation and an under-representation of the integer frequencies respectively. The model's performance in these different settings increases with the number of neurons utilized for the encoding. However, the increase in accuracy is not extremely sharp, having 88.7% with 20 neurons, 92.33% with 80, and 92.36% with 160. This indicates that the most prominent features in the data could potentially be effectively represented with fewer frequencies. In this work, the frequency set is selected linearly in the defined space, however, other ad-hoc sets of frequencies could lead to interesting results. Concerning the use of learnable delays, experiments have shown no significant difference in this work, potentially due to the use of a loss function that relies on

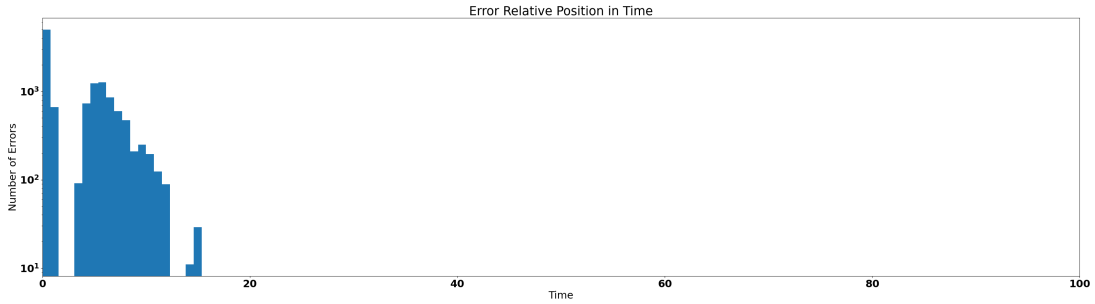


Fig. 6.7: Histogram of the relative time of the classification errors. Time is expressed as the percentage from the beginning of the action until the end. As the chart shows, most of the errors occur within the first 16% of the relative time of each action with the overall number of errors decreasing as the relative time increases. This underlines the increased difficulty posed by the onset of an action.

the overall count of spikes rather than their precise timing.

Finally, when considering EMG gestures signal classification it can be challenging to attribute a class to the onset of an action. This is because the muscular activity involved in reaching a certain position can be different from that involved in holding it [285], so much so that they are often assigned a task of their own in practical implementations. In this work, the onset is not given a separate class, however, further investigations were performed to understand where in the signal the SNN would make the most mistakes. Fig. 6.7 reports the mistake counts as a progression from 0% of the signal to 100% of it. Interestingly, most of the errors seem to in fact occur within the first 16% of the action duration, which mostly corresponds to the moment the action is initiated.

6.6 Conclusions

In this chapter, a novel approach to real-time Electromyography gestures signal classification was presented. The new approach leverages the power of the Resonate-and-Fire spiking neurons along with the possibility of implementing graded spikes in recent NM technologies. RF neurons are used to encode EMG signals in a spike-frequency domain whilst maintaining high temporal resolution. Furthermore, the so-designed encoding layer acts as a passive band-pass filter, effectively leaving out unwanted frequency bands at no extra computational cost. Due to the number of hyper-parameters necessary to operate the RF neurons layer, a system to decode a RF-encoded signal was devised as a means to facilitate the evaluation of the quality of the spike-frequency representation by comparing the decoded signal with the original signal. This in turn allowed for

automatic hyper-parameter optimization which, in combination with manual tuning, allowed a meaningful encoding of the data. This was further validated by the performed experiments. Despite the use of an extremely challenging dataset, the designed SNNs trained with the so-encoded data were in fact able to reach high levels of validation accuracy, thereby performing better than a number of other works found in the literature. The chapter also explores the impact of varying some of the parameters in the pipeline on the overall performance, and it is found that while having a higher number of encoding neurons leads to higher levels of accuracy, a lower number of them still allows for achieving performance levels that are comparable if not superior to the state-of-the-art. This is an interesting result given that the set of frequencies represented with the encoding layer in this work was obtained by mere linear sampling. If a specific set of frequencies of interest were to be selected, this could potentially close the gap in accuracy, thus allowing the same high-level performance with four times fewer encoding neurons (and hidden units in the SNN). To conclude, this chapter introduced a Neuromorphic EMG classification system that surpasses existing methodologies in accuracy, whilst retaining real-time feasibility, low-power consumption and low latency. This showcased how the use of the right spiking neuron model, the Resonate-and-Fire model in this case, to process data can be beneficial to NM solutions, and sets the ground for interesting future works, where the system could be applied to real embedded use cases, potentially representing a turning point in different fields such as robotic arm control and prosthetics.

Chapter 7

Conclusions and Future Directions

This thesis has investigated the capabilities and applications of Spiking Neural Networks, thereby contributing to the advancement of the understanding of Neuromorphic computing paradigms. The research presents a path that begins with the exploration of different models of spiking neurons, passing through the importance of employing specific neuron models as an encoding system capable of extracting important features from the data, to the exploration of less conventional applications for the SNNs, ultimately demonstrating the ability of SNNs to outperform existing systems whilst maintaining the *Neuromorphic advantage*.

Chapter 4 examines the behaviour of various spiking neuron models, namely the LIF, EIF and QIF, in response to event-based data. The study, instigated by the fact that the LIF represents the de facto standard choice despite the large variety of models present in literature, reveals that while simple models like LIF neurons can achieve satisfactory performance, more complex models such as EIF and QIF neurons can provide enhanced sensitivity and accuracy, especially in tasks that demand high temporal precision. This is demonstrated within the context of a relatively simple system, so as to allow stronger conclusions to be drawn in relation to the neuron models themselves. Thanks to this study, it has been possible to ascertain the need for a careful choice of spiking neurons when designing an NM system. In fact, this is often overlooked as a simple matter of merely using a spiking neuron model for a neural network to be considered an SNN, or as dictated by the specific hardware at hand. However, evidence shows that different types of neurons can be more or less advantageous with respect to resolving different types of tasks and that, by enabling this level of customization, NM devices would become more versatile and adaptable across a

wider range of applications. This is further testified by the recent developments in Loihi 2, which does in fact allow spiking neuron customization. The study, nonetheless, has its limitations. In order to ensure a fair comparison, only a restricted set of single-variable neuron models is analyzed. This is to show how different levels of complexity in the models can lead to different results despite merely changing the dynamics of a single variable. However, far more complex and variegated models are present in the literature that are also employed from time to time. Such models could enable even further enhancements and feature extraction capabilities. Furthermore, this study utilizes STDP as a learning rule. The reason for this lies in the fact that STDP enables weight learning solely and directly as the result of the spikes emitted by each neuron, and not as the result of an overarching optimization algorithm. However, backpropagation algorithms are now widely used in the field and could result in a completely different set of weights being learnt.

In Chapter 5, the focus shifts to the development of a novel time series forecasting approach using differencing spike encoding in conjunction with SNNs. This encoding method leverages concepts from event vision sensors and classical time series analysis to render time series more stationary and thus easier to process and learn. It is shown that the proposed encoding system approximates the encoding of the derivative of the incoming signal, thereby capturing dynamic changes in input data. The chapter also presents two novel loss functions for the SLAYER learning rule. The empirical results employing the encoding system and learning rules demonstrate that this approach improves forecasting accuracy with respect to the vanilla SLAYER approach, as well as conventional time series forecasting systems based on SARIMA. This demonstrates that a shift in the conceptual design of the encoding system can prove beneficial. Encoding systems are the means by which non-NM applications can be approached with the use of SNNs. By all means, they are substitutes for NM sensors, which do not yet exist in certain contexts such as time series forecasting ones. As such, they should be thought of as a way to sense data in a way that can benefit its learning from the SNN's point of view by design. Time series are possibly one of the data types that undergo the most ample pre-processing to make them more stationary, normalize them and extract certain features. These processes do not normally fit within an NM processing pipeline as they are (and thus require a conventional CPU to perform them), but could be instead integrated as part of the encoding by designing populations of neurons that react in certain ways. As a matter of fact, this has been shown to not only be possible, but to also en-

able achieving high-performance levels. Nonetheless, the approach presented in this chapter has some limitations. The presented encoding system incorporates the concept of differencing from classical time series analysis. This is but one of the ways in which time series are pre-processed, and the inclusion of yet other systems can be conceived. Furthermore, the DecodingLoss presented here relies heavily on the use of the presented encoding system, and could only otherwise be used by defining a set of arbitrary decoding value ranges, which would potentially not be directly relatable to the input. Finally, a disadvantage of this study is in performing next-time-step predictions only. While it is possible to maintain the prediction stream for prolonged series, further step-ahead predictions could be of interest in certain contexts.

Chapter 6 further explores the practical applications of SNNs by addressing the classification of EMG signals for gesture recognition. The proposed Resonate-and-Fire neuron encoding scheme performs encoding, filtering and feature extraction all at once at no extra cost. This is thanks to the RF neurons' ability to encode frequency information and, for a population of them, to behave in a manner similar to a Short Time Fourier Transform. The chapter further proposes a novel decoding system based on convolutional layers that can translate a spike-frequency-encoded signal back into the temporal domain. This decoding system is then utilized to perform hyperparameter optimization and thus select a set of hyperparameters that is (heuristically) optimal for the encoding of the EMG gesture signals. The proposed pipeline is validated on an extremely challenging dataset, and the results show that this method achieves high classification accuracy despite high class imbalance and low inter-class variance, outperforming existing methodologies in the field. This chapter provides concrete evidence of the efficacy of SNNs combined with suitable spiking neuron-based encoding schemes in practical, real-world applications, reinforcing the conclusions drawn from the previous chapters. When compared to similar works in the literature, the proposed method has in fact the potential to tackle some key points raised in Chapter 3 (see 3.4) relative to competitive accuracy, real-time processing, and need for data pre-processing. By employing RF neurons for the encoding, the method not only enables an end-to-end NM approach but also performs useful filtering and extraction of frequency-based features as part of the encoding step, which has been discussed to be beneficial in the Chapter 5. The overall solution utilizes a fully connected SNN architecture with only three layers, in a comparable way to other works in the literature that reported low energy consumption. The use of small time windows with an 80% overlap makes the solution affine to real-time

processing and real-world implementations. The performance level achieved overall places the proposed pipeline amongst the best-performing ones in the state of the art. For the reasons above, EMG gesture classification is deemable as a prime avenue for NM applications, having shown that a high level of accuracy can be attained whilst maintaining the advantages brought by NM technologies. Nevertheless, it is worthwhile to point out some of the limitations that the study may incur. Existing sEMG sensing devices perceive muscular activity as a stream of continuous/discrete values at a certain sampling rate. A Neuromorphic sEMG sensing device does not therefore exist yet that could implement an encoding system based on the proposed RF neuron population, thus limiting the practical applicability to, for instance, a solution employing an NM chip mounted on a more conventional board (with the annexed computational overhead derived from buffering, synchronization, and so on). Furthermore, the proposed solution is based on the use of 8 or 16 sEMG sensors (channels). However, for certain types of use cases relating to prosthetics for limb replacement, it is not always possible to fit this number of sensors, and the solution would therefore need to be further tuned to ensure the same level of performance with a lower number of electrodes.

In conclusion, this thesis has approached the NM field with a bottom-up approach. Investigations begin with questions about the functionalities of spiking neurons, which are a core component of SNNs. This initial work informs on the advantages of accurately selecting neuron model depending and their functionality and on the task at hand. This evolves into research on the use of spiking neurons as an information-extracting encoding step for time series forecasting, demonstrating how this practice can be beneficial to obtain free pre-processing on the data. Finally, gathering the teachings learnt from the previous steps, the work delves into the design of a state-of-the-art SNN system, which ties in ad-hoc spiking neurons, the RF neurons, for sEMG signal encoding, filtering, and classification.

While each individual piece of work has focused on bounded case scenarios, the results obtained and their implications extend beyond the boundaries of the experiments. Spiking neurons are what differentiates an SNN from a common ANN. Despite the focus of Chapter 4 being on three simple neuron models, considerations can be expanded to any other model. In particular the importance of determining which typology of neuron better fits a particular task, and thus the importance for hardware components (NM chips) to be able to support a variety of models, so as to allow for all-round experimentations of NM pipelines,

and to enable a larger pool of applications to be served by NM solutions. Time series are a concept that sit at the foundations of any type of task that involves time-indexed data. Aside from the specific energy load forecasting applications discussed here, the research work is, in principle, data agnostic and can thus be considered foundational for a breadth of other time-series-based problems. A particular case is represented by sequence analysis such as natural language processing. This work approaches time series with an all-new conceptual approach, where spiking neuron-based encoding is at its core. Developments based on this conceptualization can bring to more energy efficient, and yet still competitive alternatives to LLMs. The implications on a wider scope of the work on sEMG classification step from the promising results that the developed pipeline has shown. The use of ad-hoc spiking neuron-based encoding mechanisms, as a matter of fact, affects not only the possibility of using SNNs to engage data that is not normally considered NM, but also the potential for developments in the NM sensing department. Through the results of this work, the RF neuron-based encoding is in fact shown to be a valid candidate for an sEMG NM sensor implementation, where information about muscular activity is directly sensed and transmitted by RF neurons. As whole, this work is a demonstration of how SNNs are closing down on the gap with conventional ANNs and, in some cases, are even surpassing it. The advancements in the NM computing research are proceeding at high velocity and building on top of one another. This rapid growth will soon enable NM solutions to find their place in state-of-the-art approaches not only delivering at the promised energy efficiency level, but also at the performance level.

7.1 Future Directions

Looking ahead, there are several promising avenues for future research that naturally emerge from the findings and insights gained through the research work presented in this thesis. These potential directions offer opportunities to deepen the understanding of the subject matter, explore new dimensions of the topic, and address any limitations or open questions identified in the current work. By building on the foundation laid by this thesis, future research can further contribute to the advancement of knowledge in this field, potentially leading to innovative applications and broader theoretical developments.

Analysis of Spiking Neurons. The results and the related discussions in Chapter 4 highlighted the importance of the choice of the neuron models within an SNN. In particular, they showed, in principle, how neurons embedding more or less complex internal dynamics could be more suitable for types of data that are composed of subtle temporal features. Future works in this direction can focus on the understanding of the interplay between neural network depth and different types of neuron models in the extraction and learning of complex spatiotemporal features in such data. Furthermore, interesting insights may result from investigating the use of different types of neuron models paired with backpropagation-based learning rules, as well as looking into the possibility of utilizing different types of neuron models within the same SNN.

Time Series Forecasting. The findings in Chapter 5 demonstrated that encoding mechanisms embedding pre-processing concepts can be beneficial to developing SNNs for time series forecasting. Interesting future directions stemming from this piece of research can look at embedding different types of pre-processing, perhaps looking at seasonal differencing and second-order differencing, into the spike encoding mechanism. Furthermore, they can regard the investigation into using diverse types of time series, including multi-variate time series, as well as approaching multiple time step-ahead forecasting.

EMG Gesture Classification. Chapter 6 demonstrated that SNNs can prove more effective than conventional methods given the right setting and spiking neuron model selection. Given the success of the proposed approach, future works can look at hardware implementations in combination with existing sEMG devices, as a means to validate the method's suitability for real, impactful applications. Furthermore, investigating the possibility of utilizing a very low number of EMG sensors, as well as seeking to design a prototype of an RF-based NM EMG sensor, promises to be extremely interesting and useful developments.

Appendix A

Extension to the SpykeTorch Framework

For the purpose of developing NM-ML algorithms based on STDP, SpykeTorch allows a high degree of customization and flexibility to the user. However, as mentioned in 2.2.5.1.4, the framework originally provides a single spiking neuron model, the IF. This model does not have a voltage leakage factor, which means that its internal state can only increase until it is reset. In order to accommodate the need to test a variety of spiking neurons and hence augment the usage potential of SpykeTorch, an expansion to the library is proposed that implements a new set of eight spiking neuron models as shown in Table A.1. By introducing more complex neuron models, the original workflow and implementation patterns adopted in the original framework cannot be easily utilized. Therefore, some details about the differences that are introduced to accommodate such neuron models in the library are here discussed. The framework resulting from the applied changes is referred to as SpykeTorch-Extended.

A.1 Implementation Details

Given their computational efficiency, the family of phenomenological spiking neuron models is well suited for large-scale neural networks in the context of machine learning. Because of this, the proposed expansion includes a subset of this family of neurons, namely:

- Leaky Integrate-and-Fire [70],
- Exponential Integrate-and-Fire [77],
- Quadratic Integrate-and-Fire [79],

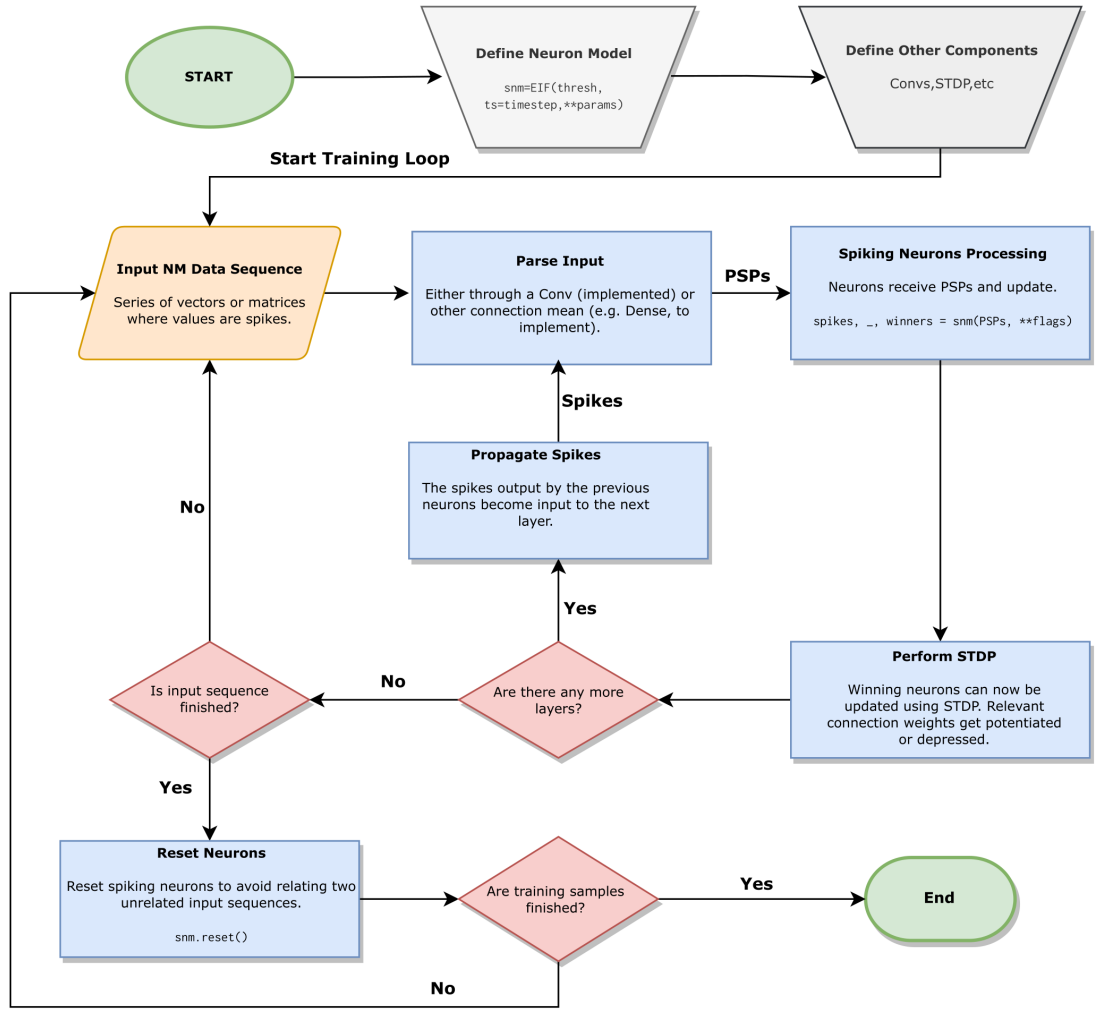


Fig. A.1: Example flowchart for SpykeTorch-Extended. After the definition of the components of the SNN, each data sample is required to be decomposed into its forming time steps before being processed by the SNN. This ensures that learnt parameters will influence the result of the next iteration.

- Adaptive Exponential Integrate-and-Fire [78],
- Izhikevich's [16],
- Heterogeneous Neurons (LIF, QIF, QIF)

Due to the greater complexity of the newly introduced neurons, a deviation from the original implementation of the framework is required. The expansion thus adopts an object-oriented approach for the neurons, which allows them to retain an internal state and other properties. Nevertheless, to maintain compatibility, neuron objects are callable (i.e. utilizable like a function) and share the same output format as in the original version. Furthermore, neurons are not restricted to firing only once per input sequence. This solely depends on the choice

of parameters for a given neuron, such as the refractory period. Another difference with the previous implementation is that neurons are expected to receive and process events one time-step at a time. While this introduces overhead on the computational time, it has a number of advantages. Single time-step processing allows simulating real-time; at the same time, it ensures that the decay of the membrane potential can happen before the next input is received; finally, it allows for weight updates due to STDP to be enacted between each consecutive input, thus allowing them to affect every subsequent moment in time and making the system more realistic. From a more practical point of view, a neuron layer in SpykeTorch-Extended is characterized by at least the set of parameters of a LIF neuron; however, more complex neuron models will require more parameters. A layer of neurons in this system can be better depicted as a set of neuronal populations. The number and size of the population reflect that of the input that is processed by the layer. Let's consider the example where the synaptic connection between two layers in an SNN is given by a convolutional layer. The convolution produces a set of C feature maps of size H x W, where every element represents the post-synaptic potential (PSP) that is going to flow into a neuron on the receiving end. In this setting, each feature map c_i with $i \in \{0, \ldots, C-1\}$ represents a population of $N = H \cdot W$ neurons that share the same set of weights (the kernel of the convolution). Therefore, each neuron in this population will be receptive to the same feature in the input, but each will encode a different spatio-temporal position for it.

As a result of the changes above, the standard workflow in SpykeTorch-Extended was adjusted with respect to the original version. In Figure A.1, an example flowchart of a pipeline using the new neuron models is reported. As the flowchart highlights, each input is expected to be unravelled into all the time steps it is composed of and, for each time step, all the events that took place in such a time span are to be fed forward to the SNN.

A.1.1 LIF Neuron Class

The `LIF` neuron class is the implementation the homonym neuron model. A LIF neuron layer is characterized by the following parameters: a time constant `tau_rc`, a capacitance `C`, a time-step size `ts`, a `threshold`, a `resting_potential`, and by the number of `refractory_timesteps`. The same parameters are present also the other classes of neurons. Neurons in this layer perform membrane updates according to the solution of the differential equation presented in [12].

Tab. A.1. Summary of newly added spiking neurons to SpykeTorch. All the neurons share a base set of parameters with the LIF, but they may require more depending on the neuron type, which are briefly reported in the short description.

Neurons	Short Description
LIF [70]	Uses the integral solution to the differential equation in [12].
EIF [77]	Single-variable model with an exponential dependency. Has parameters `delta_t` for the sharpness of the curve, and `theta_rh` as a cut-off threshold for the upswing of the curve [12].
QIF [79]	Single-variable model with a quadratic dependency. Has parameters `a` for the steepness of the quadratic curve, and `u_c` as the negative-to-positive updates crossing point of the membrane potential [12].
AdEx [78]	Two-variables model similar to the EIF, but with an adaptation variable. It adds parameters `a` and `b`, respectively for adaptation-potential coupling and adaptation increase upon spike emission.
IZ [16]	Two-variables model similar to the QIF, but with an adaptation variable. It adds parameters `a` for the time scale of the adaptation variable, `b` for the sub-threshold sensitivity of the adaptation, and `d` for the adaptation increase upon spike emission.
H-Neurons	Heterogeneous versions of LIF, EIF, and QIF neurons with uniformly distributed `tau_rc` parameter.

A.1.2 EIF Neuron Class

The `EIF` class implements the EIF definition in [12]. It takes two further parameters than the `LIF` class, namely `delta_t` and `theta_rh`. The former defines the sharpness of the exponential function, the latter determines the membrane potential value where the function begins its upswing. If not given, `theta_rh` is estimated as being $\frac{3}{4}$ of the firing threshold.

A.1.3 QIF Neuron Class

This class implement the QIF model as described in [12]. Like the EIF, it accepts two further parameters with respect to the LIF, but these are `a` and `u_c`. The former regulates the steepness of the quadratic curve, whereas the latter determines the crossing point with the x-axis, hence the value of the membrane potential after which the next updates will facilitate its increase even in the absence of an input. If not given, `u_c` is estimated as being $\frac{3}{4}$ of the firing threshold.

A.1.4 AdEx Neuron Class

The model implemented by this class is the homonym described in [78]. Compared to the EIF implementation, it accepts two further parameters, namely `a`, which determines the strength of the coupling of the adaptation variable with the membrane potential, and `b`, which is an amount of current by which the adaptation gets increased every time a spike is fired.

A.1.5 Izhikevich's Neuron Class

The class implementation of Izhikevich's neuron model follows the differential equations presented in [16]. The class accepts three further parameters with respect to the `LIF` class. These are `a`, i.e. the time scale of the adaptation, `b`, i.e. the sensitivity to subthreshold fluctuations of the adaptation, and `d`, which is an amount of current by which the adaptation gets increased every time a spike is fired.

A.1.6 Heterogeneous Neuron Classes

The implemented neuron classes create a layer of spiking neurons that share the same hyperparameters. This is referred to as being a homogeneous layer of neurons because they all react in the same way to the same sequence of inputs. However, it might be useful to have neurons reacting differently to one input, since this could mean being able to learn different kinds of temporal patterns within the same layer. Because of this, a further group of heterogeneous neuron classes for the `LIF`, `EIF`, and `QIF` classes is developed. Specifically, they provide a set of τ_{rc} values that are uniformly distributed within a range specified by the user through the parameter `tau_range`. Being able to have a set of spiking neurons with different τ_{rc} allows sensibility to different time scales. The limitation to uniform distributions was chosen to maintain simplicity, however, this can be easily extended to custom distributions.

Bibliography

[1] D. M. Dimiduk, E. A. Holm, and S. R. Niezgoda, "Perspectives on the impact of machine learning, deep learning, and artificial intelligence on materials, processes, and structures engineering," *Integrating Materials and Manufacturing Innovation*, vol. 7, no. 3, pp. 157–172, Aug. 2018. 1

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. 1, 8

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: http://dx.doi.org/10.1145/3065386 1, 8, 10

[4] K. Bayoudh, R. Knani, F. Hamdaoui, and A. Mtibaa, "A survey on deep multimodal learning for computer vision: advances, trends, applications, and datasets," *The Visual Computer*, vol. 38, pp. 2939–2970, 2021. 1

[5] M. Taye, "Understanding of machine learning with deep learning: architectures, workflow, applications and future directions," *Computers*, vol. 12, p. 91, 2023. 1

[6] R. Adebayo, "Ai-enhanced manufacturing robotics: a review of applications and trends," *World Journal of Advanced Research and Reviews*, vol. 21, pp. 2060–2072, 2023. 1

[7] M. Shafique, T. Theocharides, C.-S. Bouganis, M. A. Hanif, F. Khalid, R. Hafiz, and S. Rehman, "An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the iot era," in *2018 Design, Automation amp; Test in Europe Conference amp; Exhibition (DATE)*. IEEE, Mar. 2018. 1, 3

[8] H. Atlam, R. Walters, and G. Wills, "Intelligence of things: opportunities amp; challenges," 2018. 2

[9] A. Marengo, "The future of ai in iot: emerging trends in intelligent data analysis and privacy protection," 2023. 2

[10] D. V. Christensen, R. Dittmann, B. Linares-Barranco, A. Sebastian, M. Le Gallo, A. Redaelli, S. Slesazeck, T. Mikolajick, S. Spiga, S. Menzel, I. Valov, G. Milano, C. Ricciardi, S.-J. Liang, F. Miao, M. Lanza, T. J. Quill, S. T. Keene, A. Salleo, J. Grollier, D. Marković, A. Mizrahi, P. Yao, J. J. Yang, G. Indiveri, J. P. Strachan, S. Datta, E. Vianello, A. Valentian, J. Feldmann, X. Li, W. H. P. Pernice, H. Bhaskaran, S. Furber, E. Neftci, F. Scherr, W. Maass, S. Ramaswamy, J. Tapson, P. Panda, Y. Kim, G. Tanaka, S. Thorpe, C. Bartolozzi, T. A. Cleland, C. Posch, S. Liu, G. Panuccio, M. Mahmud, A. N. Mazumder, M. Hosseini, T. Mohsenin, E. Donati, S. Tolu, R. Galeazzi, M. E. Christensen, S. Holm, D. Ielmini, and N. Pryds, "2022 roadmap on neuromorphic computing and engineering," *Neuromorphic Computing and Engineering*, vol. 2, no. 2, p. 022501, may 2022. 2, 3, 4, 19, 35, 53, 64, 69, 98

[11] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, vol. 10, 2020. 2, 66

[12] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics*. Cambridge University Press, 2009. 2, 3, 20, 21, 23, 25, 28, 29, 30, 33, 37, 48, 50, 77, 80, 81, 92, 101, 147, 148, 149

[13] K. Yamazaki, V.-K. Vo-Ho, D. Bulsara, and N. Le, "Spiking neural networks and their applications: A review," *Brain Sciences*, vol. 12, no. 7, p. 863, Jun. 2022. 2

[14] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, oct 2018. 2

[15] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, jan 2022. 2, 3, 45

[16] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, nov 2003. 3, 25, 33, 47, 81, 146, 148, 149

[17] D. L. Manna, A. Vicente-Sola, P. Kirkland, T. Bihl, and G. Di Caterina, "Simple and complex spiking neurons: perspectives and analysis in a simple stdp scenario," *Neuromorphic Computing and Engineering*, 2022. 3, 45, 99, 100

[18] A. Vicente-Sola, D. L. Manna, P. Kirkland, G. D. Caterina, and T. Bihl, "Evaluating the temporal understanding of neural networks on event-based action recognition with dvs-gesture-chain," Sep. 2022. 3, 79, 98

[19] E. P. Frady and F. T. Sommer, "Robust computation with rhythmic spike patterns," *Proceedings of the National Academy of Sciences*, vol. 116, no. 36, pp. 18 050–18 059, Aug. 2019. 4, 24, 32

[20] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Frontiers in Neuroscience*, vol. 13, 2019. 5, 45, 51

[21] E. Aguilar Madrid and N. Antonio, "Short-term electricity load forecasting with machine learning," *Information*, vol. 12, no. 2, p. 50, jan 2021. 5, 62, 63, 102

[22] B. Ibrahim and L. Rabelo, "A deep learning approach for peak load forecasting: A case study on panama," *Energies*, vol. 14, no. 11, p. 3039, May 2021. 103

[23] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*, vol. 35, no. 12. AAAI Press, 2021, pp. 11 106–11 115. 5, 63, 102

[24] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 1419–1428. 5, 41, 48, 101, 108, 123, 131

[25] S. Pizzolato, L. Tagliapietra, M. Cognolato, M. Reggiani, H. Müller, and M. Atzori, "Comparison of six electromyography acquisition setups on hand movement classification tasks," *PLOS ONE*, vol. 12, no. 10, p. e0186132, Oct. 2017. [Online]. Available: http://dx.doi.org/10.1371/journal.pone.0186132 5, 53, 73, 123, 124, 125

[26] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, p. 115–133, Dec. 1943. [Online]. Available: http://dx.doi.org/10.1007/BF02478259 8, 9, 19

[27] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015. 8

[28] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974. 8

[29] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," 2006. 8

[30] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 873–880. 8

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034. 8

[32] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958. 9, 19

[33] 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. 9, 14

[34] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: http://ieeexplore.ieee.org/document/726791/ 10

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org. 10, 11, 13, 16, 96

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2014. 10

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 10

[38] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017. 11

[39] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 2146–2153. 12

[40] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," 2013. 12

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015. 12

[42] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv: Learning*, 2016. 12

[43] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *ArXiv*, vol. abs/1710.05941, 2018. 12

[44] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: comparison of trends in practice and research for deep learning," Jan. 2021, pp. 124 – 133, 2nd International Conference on Computational Sciences and Technology, (INCCST) ; Conference date: 17-12-2020 Through 19-12-2020. 12

[45] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on mnist classification task," *arXiv preprint arXiv:1804.02763*, 2018. 12

[46] S. Eger, P. Youssef, and I. Gurevych, "Is it time to swish? comparing deep learning activation functions across nlp tasks," *arXiv preprint arXiv:1901.02671*, 2019.

[47] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 1836–1841.

[48] Q. Zheng, M. Yang, X. Tian, X. Wang, and D. Wang, "Rethinking the role of activation functions in deep convolutional neural networks for image classification." *Engineering Letters*, vol. 28, no. 1, 2020.

[49] M. Goyal, R. Goyal, P. Venkatappa Reddy, and B. Lall, *Activation Functions*. Cham: Springer International Publishing, 2020, pp. 1–30.

[50] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," *Journal of Computational Physics*, vol. 404, p. 109136, 2020. 12

[51] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012. 12

[52] X. Wang, X. Lin, and X. Dang, "Supervised learning in spiking neural networks: A review of algorithms and evaluations," *Neural Networks*, vol. 125, pp. 258–280, may 2020. 14

[53] K. Fukushima, "Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/7370364 19

[54] C. Mead, *Analog VLSI and neural systems*. Addison-Wesley Longman Publishing Co., Inc., 1989. [Online]. Available: https://dl.acm.org/citation.cfm?id=64998 19

[55] S. Choudhary, S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," in *International conference on artificial neural networks*. Springer, 2012, pp. 121–128. 19

[56] E. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, sep 2004. 19, 23, 27, 28, 29, 34

[57] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Science*, vol. 338, no. 6111, pp. 1202–1205, nov 2012. 19

[58] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network." *Frontiers in neuroscience*, vol. 7, p. 178, 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24115919http://www.ncbi.nlm.nih.gov/pubmed/24115919 19

[59] P. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, 2015. 84

[60] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in neural information processing systems*, 2015, pp. 1117–1125.

[61] E. Hunsberger and C. Eliasmith, "Spiking deep networks with lif neurons," Oct. 2015. 29

[62] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, dec 2017. 19, 48

[63] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück, "Steering a predator robot using a mixed frame/event-driven convolutional neural network," in *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. IEEE, 2016, pp. 1–8. 19

[64] I.-A. Lungu, F. Corradi, and T. Delbrück, "Live demonstration: Convolutional neural network driven by dynamic vision sensor playing roshambo," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–1. 19

[65] L. Wang, Y.-S. Ho, K.-J. Yoon *et al.*, "Event-based high dynamic range image and very high frame rate video generation using conditional generative

adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 081–10 090. 19

[66] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 19

[67] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Ev-flownet: Self-supervised optical flow estimation for event-based cameras," *arXiv preprint arXiv:1802.06898*, 2018. 19

[68] ——, "Unsupervised event-based learning of optical flow, depth, and egomotion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 989–997. 19

[69] L. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, no. 5-6, pp. 303–304, nov 1999. 23

[70] L. Lapicque, "Recherches quantitatives sur l'excitation electrique des nerfs traitée comme une polarization," *Journal de Physiologie et Pathologie General*, vol. 9, pp. 620–635, 1907. 23, 29, 47, 77, 108, 145, 148

[71] M. J. E. Richardson, N. Brunel, and V. Hakim, "From subthreshold to firing-rate resonance," *Journal of Neurophysiology*, vol. 89, no. 5, pp. 2538–2554, may 2003. 23

[72] G. D. Smith, C. L. Cox, S. M. Sherman, and J. Rinzel, "Fourier analysis of sinusoidally driven thalamocortical relay neurons and a minimal integrate-and-fire-or-burst model," *Journal of Neurophysiology*, vol. 83, no. 1, pp. 588–610, jan 2000. 23

[73] Wondimu W Teka, "The fractional order leaky integrate-and-fire model: Fractional differentiation-spiking properties," 2015. 23

[74] S. Schliebs, A. Mohemmed, and N. Kasabov, "Are probabilistic spiking neural networks suitable for reservoir computing?" in *The 2011 International Joint Conference on Neural Networks*. IEEE, jul 2011. 24

[75] E. M. Izhikevich, "Resonate-and-fire neurons," *Neural Networks*, vol. 14, no. 6-7, pp. 883–894, jul 2001. 24, 32, 48

[76] E. P. Frady, S. Sanborn, S. B. Shrestha, D. B. D. Rubin, G. Orchard, F. T. Sommer, and M. Davies, "Efficient neuromorphic signal processing with resonator neurons," *Journal of Signal Processing Systems*, vol. 94, no. 10, pp. 917–927, may 2022. 24, 32, 45, 108, 123, 126

[77] N. Fourcaud-Trocmé, D. Hansel, C. van Vreeswijk, and N. Brunel, "How spike generation mechanisms determine the neuronal response to fluctuating inputs," *The Journal of Neuroscience*, vol. 23, no. 37, pp. 11 628–11 640, dec 2003. 24, 30, 50, 145, 148

[78] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of Neurophysiology*, vol. 94, no. 5, pp. 3637–3642, nov 2005. 24, 33, 146, 148, 149

[79] G. B. Ermentrout and N. Kopell, "Parabolic bursting in an excitable system coupled with a slow oscillation," *SIAM Journal on Applied Mathematics*, vol. 46, no. 2, pp. 233–253, apr 1986. 25, 30, 145, 148

[80] J. Touboul, "Bifurcation analysis of a general class of nonlinear integrate-and-fire neurons," *SIAM Journal on Applied Mathematics*, vol. 68, no. 4, pp. 1045–1079, jan 2008. 25

[81] N. Kasabov, "To spike or not to spike: A probabilistic spiking neuron model," *Neural Networks*, vol. 23, no. 1, pp. 16–19, jan 2010. 25

[82] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical Journal*, vol. 1, no. 6, pp. 445–466, jul 1961. 26, 28

[83] J. Nagumo, S. Arimoto, and S. Yoshizawa, "An active pulse transmission line simulating nerve axon," *Proceedings of the IRE*, vol. 50, no. 10, pp. 2061–2070, oct 1962. 26, 28

[84] J. L. Hindmarsh and R. M. Rose, "A model of neuronal bursting using three coupled first order differential equations," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 221, no. 1222, pp. 87–102, mar 1984. 26, 28

[85] C. Morris and H. Lecar, "Voltage oscillations in the barnacle giant muscle fiber," *Biophysical Journal*, vol. 35, no. 1, pp. 193–213, jul 1981. 26, 28

[86] 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, pp. 500–544, aug 1952. 26, 28

[87] L. L. Neves and L. H. A. Monteiro, "A linear analysis of coupled wilson-cowan neuronal populations," *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1–6, 2016. 27

[88] H. R. Wilson and J. D. Cowan, "Excitatory and inhibitory interactions in localized populations of model neurons," *Biophysical Journal*, vol. 12, no. 1, pp. 1–24, jan 1972. 27

[89] E. M. Izhikevich, *Dynamical Systems in Neuroscience*. MIT Press Ltd, Jan. 2010. 28, 30, 81

[90] S. A. Aamir, Y. Stradmann, P. Muller, C. Pehle, A. Hartel, A. Grubl, J. Schemmel, and K. Meier, "An accelerated LIF neuronal network array for a large-scale mixed-signal neuromorphic architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4299–4312, dec 2018. 29

[91] A. Diamond, T. Nowotny, and M. Schmuker, "Comparing neuromorphic solutions in action: Implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms," *Frontiers in Neuroscience*, vol. 9, jan 2016.

[92] K. E. Friedl, A. R. Voelker, A. Peer, and C. Eliasmith, "Human-inspired neurorobotic system for classifying surface textures by touch," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 516–523, jan 2016.

[93] J. Göltz, L. Kriener, A. Baumbach, S. Billaudelle, O. Breitwieser, B. Cramer, D. Dold, A. F. Kungl, W. Senn, J. Schemmel, K. Meier, and M. A. Petrovici, "Fast and deep: energy-efficient neuromorphic learning with first-spike times," Dec. 2019.

[94] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated STDP," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6178–6190, dec 2018.

[95] E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.-C. Liu, and S. Furber, "Scalable energy-efficient, low-latency implementations of trained spiking

deep belief networks on SpiNNaker," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2015.

[96] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," Jun. 2016. 40, 41

[97] S. Chaturvedi, A. Khurshid, and M. Boudjelal, "Image segmentation using leaky integrate-and-fire model of spiking neural network," *International Journal of Wisdom Based Computing*, vol. 2, no. 1, pp. 21–28, 2012.

[98] C. Jiang, L. Yang, and Y. Zhang, "A spiking neural network with spike-timing-dependent plasticity for surface roughness analysis," *IEEE Sensors Journal*, vol. 22, no. 1, pp. 438–445, 2021.

[99] A. Patino-Saucedo, H. Rostro-González, T. Serrano-Gotarredona, and B. Linares-Barranco, "Event-driven implementation of deep spiking convolutional neural networks for supervised classification using the spinnaker neuromorphic platform," *Neural networks : the official journal of the International Neural Network Society*, vol. 121, pp. 319–328, 2020. 29

[100] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, no. 2, p. 1–35, Apr. 2019. [Online]. Available: http://dx.doi.org/10.1145/3304103 29, 30

[101] G. Orchard, E. P. Frady, D. B. D. Rubin, S. Sanborn, S. B. Shrestha, F. T. Sommer, and M. Davies, "Efficient neuromorphic signal processing with loihi 2," in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2021, pp. 254–259. 32, 43, 45, 78, 99

[102] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. El Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: A review of tools and strategies," *Journal of Computational Neuroscience*, vol. 23, no. 3, p. 349–398, Jul. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10827-007-0038-6 33

[103] L. Long and G. Fang, "A review of biologically plausible neuron models for spiking neural networks," in *AIAA Infotech@Aerospace 2010*. American Institute of Aeronautics and Astronautics, apr 2010. 34

[104] A. Barton, E. Volna, and M. Kotyrba, "The application perspective of izhikevich spiking neural model – the initial experimental study," in *Recent Advances in Soft Computing*. Springer International Publishing, aug 2018, pp. 223–232. 34

[105] G. Kumar, V. Aggarwal, N. Thakor, M. Schieber, and M. Kothare, "Optimal parameter estimation of the izhikevich single neuron model using experimental inter-spike interval (isi) data," 08 2010, pp. 3586 – 3591. 35

[106] C. Teeter, R. Iyer, V. Menon, N. Gouwens, D. Feng, J. Berg, A. Szafer, N. Cain, H. Zeng, M. Hawrylycz, C. Koch, and S. Mihalas, "Generalized leaky integrate-and-fire models classify multiple neuron types," *Nature Communications*, vol. 9, no. 1, feb 2018. 35

[107] R. Jolivet, A. Rauch, H.-r. Lüscher, and W. Gerstner, "Integrate-and-fire models with adaptation are good enough," in *Advances in Neural Information Processing Systems*, Y. Weiss, B. Schölkopf, and J. Platt, Eds., vol. 18. MIT Press, 2005. 35

[108] B. J. Grzyb, E. Chinellato, G. M. Wojcik, and W. A. Kaminski, "Which model to use for the liquid state machine?" in *2009 International Joint Conference on Neural Networks*. IEEE, jun 2009. 35

[109] D. H. Perkel and T. H. Bullock, "Neural coding." *Neurosciences Research Program Bulletin*, 1968. 36

[110] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex," *Neural computation*, vol. 13, no. 6, pp. 1255–1283, 2001.

[111] A. Kumar, S. Rotter, and A. Aertsen, "Spiking activity propagation in neuronal networks: reconciling different perspectives on neural coding," *Nature reviews neuroscience*, vol. 11, no. 9, pp. 615–627, 2010. 36

[112] W. Gerstner, "Population dynamics of spiking neurons: fast transients, asynchronous states, and locking," *Neural computation*, vol. 12, no. 1, pp. 43–89, 2000. 37

[113] N. Brunel, F. S. Chance, N. Fourcaud, and L. Abbott, "Effects of synaptic noise and filtering on the frequency response of spiking neurons," *Physical Review Letters*, vol. 86, no. 10, p. 2186, 2001. 37

[114] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *nature*, vol. 381, no. 6582, pp. 520–522, 1996. 38

[115] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, mar 2018. 38

[116] Q. Zhou, Y. Shi, Z. Xu, R. Qu, and G. Xu, "Classifying melanoma skin lesions using convolutional spiking neural networks with unsupervised stdp learning rule," *IEEE Access*, 2020. 38

[117] J. J. Hopfield, "Pattern recognition computation using action potential timing for stimulus representation," *Nature*, vol. 376, no. 6535, pp. 33–36, 1995. 38

[118] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997. 38

[119] 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. 38

[120] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2017. 38

[121] S. R. Kheradpisheh and T. Masquelier, "S4nn: temporal backpropagation for spiking neural networks with one spike per neuron," *International Journal of Neural Systems*, vol. 30, no. 6, p. 2050027, 2020. 38

[122] S. M. Bohte, "The evidence for neural information processing with precise spike-times: A survey," *Natural Computing*, vol. 3, no. 2, pp. 195–206, 2004. 38

[123] A. Borst and F. E. Theunissen, "Information theory and neural coding," *Nature neuroscience*, vol. 2, no. 11, pp. 947–957, 1999. 38

[124] W. Maass, R. Legenstein, and H. Markram, "A new approach towards vision suggested by biologically realistic neural microcircuit models," in *International Workshop on Biologically Motivated Computer Vision*. Springer, 2002, pp. 282–293. 38

[125] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting," *Neural computation*, vol. 22, no. 2, pp. 467–510, 2010. 38, 40

[126] "Hebb, D. O. Organization of behavior. New York: Wiley, 1949, pp. 335, $4.00," *Journal of Clinical Psychology*, vol. 6, no. 3, pp. 307–307, jul 1949. [Online]. Available: http://doi.wiley.com/10.1002/1097-4679{%}28195007{%}296{%}3A3{%}3C307{%}3A{%}3AAID-JCLP2270060338{%}3E3.0.CO{%}3B2-K 39, 78

[127] A. Morrison, M. Diesmann, and W. Gerstner, "Phenomenological models of synaptic plasticity based on spike timing," *Biological cybernetics*, vol. 98, no. 6, pp. 459–478, 2008. 39

[128] G. G. Turrigiano and S. B. Nelson, "Homeostatic plasticity in the developing nervous system," *Nature reviews neuroscience*, vol. 5, no. 2, pp. 97–107, 2004.

[129] A. Artola, S. Bröcher, and W. Singer, "Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex," *Nature*, vol. 347, pp. 69–72, 1990.

[130] E. Vasilaki and M. Giugliano, "Emergence of connectivity motifs in networks of model neurons with short-and long-term plastic synapses," *PloS one*, vol. 9, no. 1, p. e84626, 2014.

[131] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, "Connectivity reflects coding: a model of voltage-based stdp with homeostasis," *Nature neuroscience*, vol. 13, no. 3, p. 344, 2010. 39

[132] L. I. Zhang, H. W. Tao, C. E. Holt, W. A. Harris, and M.-m. Poo, "A critical window for cooperation and competition among developing retinotectal synapses," *Nature*, vol. 395, no. 6697, pp. 37–44, 1998. 39

[133] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience*, vol. 3, no. 9, pp. 919–926, 2000. 39

[134] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS computational biology*, vol. 3, no. 2, p. e31, 2007. 39, 49, 83

[135] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike pattern association neuron for learning spatio-temporal spike patterns," *International journal of neural systems*, vol. 22, no. 04, p. 1250012, 2012. 40

[136] R. V. Florian, "The chronotron: A neuron that learns to fire temporally precise spike patterns," *PloS one*, vol. 7, no. 8, p. e40233, 2012. 40

[137] ——, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural computation*, vol. 19, no. 6, pp. 1468–1502, 2007. 40

[138] J. C. Tapson, G. K. Cohen, S. Afshar, K. M. Stiefel, Y. Buskila, T. J. Hamilton, and A. van Schaik, "Synthesis of neural networks for spatio-temporal spike pattern recognition and processing," *Frontiers in neuroscience*, vol. 7, p. 153, 2013. 40

[139] 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. 40

[140] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016. 40

[141] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018. 41

[142] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 1433–1443. 41

[143] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018. 40, 49

[144] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. 42

[145] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014. 42, 78, 99

[146] H. Markram, K. Meier, T. Lippert, S. Grillner, R. Frackowiak, S. Dehaene, A. Knoll, H. Sompolinsky, K. Verstreken, J. DeFelipe *et al.*, "Introducing the human brain project," *Procedia Computer Science*, vol. 7, pp. 39–42, 2011. 42

[147] C. Mayr, S. Hoeppner, and S. Furber, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning," *arXiv preprint arXiv:1911.02385*, 2019. 42

[148] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 1947–1950. 42

[149] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2018. 43

[150] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014. 43

[151] C. Eliasmith and C. Anderson, "Neural engineering: Computation," *Representation, and Dynamics in Neurobiological Systems*, 2004. 43

[152] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang,

"Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, jan 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8259423/ 43

[153] E. P. Frady, G. Orchard, D. Florey, N. Imam, R. Liu, J. Mishra, J. Tse, A. Wild, F. T. Sommer, and M. Davies, "Neuromorphic nearest-neighbor search using intel's pohoiki springs," *arXiv preprint arXiv:2004.12691*, 2020. 43

[154] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 106–122, 2017. 43

[155] C. S. Thakur, J. L. Molin, G. Cauwenberghs, G. Indiveri, K. Kumar, N. Qiao, J. Schemmel, R. Wang, E. Chicca, J. Olson Hasler *et al.*, "Large-scale neuromorphic spiking array processors: A quest to mimic the brain," *Frontiers in neuroscience*, vol. 12, p. 891, 2018. 43

[156] Q. Liu, O. Richter, C. Nielsen, S. Sheik, G. Indiveri, and N. Qiao, "Live demonstration: Face recognition on an ultra-low power event-driven convolutional neural network asic," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0. 43

[157] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. 44

[158] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 120 dB 15micro s Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4444573/ 44

[159] 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, p. 259, 2011. [Online]. Available: http://ieeexplore.ieee.org.

[160] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, pp. 2333–2341, 2014. 44, 50, 99, 100, 102

[161] B. Son, Y. Suh, S. Kim, H. Jung, J.-S. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo, Y. Roh, H. Lee, Y. Wang, I. Ovsiannikov, and H. Ryu, "4.1 A 640x480 dynamic vision sensor with a 9\(\mu\)m pixel and 300Meps address-event representation," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, feb 2017, pp. 66–67. [Online]. Available: http://ieeexplore.ieee.org/document/7870263/ 44

[162] V. Chan, S.-C. Liu, and A. van Schaik, "Aer ear: A matched silicon cochlea pair with address event representation interface," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 48–59, 2007. 44

[163] S.-C. Liu, A. van Schaik, B. A. Minch, and T. Delbruck, "Asynchronous binaural spatial audition sensor with 2 x 64 x 4 channel output," *IEEE transactions on biomedical circuits and systems*, vol. 8, no. 4, pp. 453–464, 2013. 44

[164] J. W. Gardner and P. N. Bartlett, "A brief history of electronic noses," *Sensors and Actuators B: Chemical*, vol. 18, no. 1-3, pp. 210–211, 1994. 44

[165] W. W. Lee, S. L. Kukreja, and N. V. Thakor, "Discrimination of dynamic tactile contact by temporally precise event sensing in spiking neuromorphic networks," *Frontiers in neuroscience*, vol. 11, p. 5, 2017. 44

[166] K. A. 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. 44

[167] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283. 45

[168] 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," in *NeurIPS*, 2019. 45

[169] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, nov 2014. 45

[170] Á. M. García-Vico and F. Herrera, "A preliminary analysis on software frameworks for the development of spiking neural networks," in *Lecture Notes in Computer Science*. Springer International Publishing, 2021, pp. 564–575. 45, 46, 47

[171] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "BindsNET: A machine learning-oriented spiking neural networks library in python," *Frontiers in Neuroinformatics*, vol. 12, dec 2018. 45, 50

[172] P. Qu, L. Yang, W. Zheng, and Y. Zhang, "A review of basic software for brain-inspired computing," *CCF Transactions on High Performance Computing*, mar 2022. 46

[173] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in Neuroinformatics*, vol. 7, 2014. 47

[174] T. C. Stewart, "A technical overview of the neural engineering framework," Tech. Rep., 2012. 47

[175] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982. 47

[176] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex," *Journal of Neuroscience*, vol. 2, no. 1, pp. 32–48, 1982. 47

[177] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. de Almeida, B. McFee, H. Weideman, G. Takács, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, and J. Degrave, "Lasagne: First release." Aug. 2015. 48

[178] A. P. Davison, "PyNN: a common interface for neuronal network simulators," *Frontiers in Neuroinformatics*, vol. 2, 2008. 48

[179] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, aug 2019. 48

[180] B. Linares-Barranco, "Modular event-driven growing asynchronous simulator (megasim)," https://bitbucket.org/bernabelinares/megasim, 2018. 48

[181] *SpiNNaker: A Spiking Neural Network Architecture*. now publishers, Inc., 2020. 48

[182] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018. 48, 78, 99, 123

[183] "Lava: A software framework for neuromorphic computing," https://github.com/lava-nc/lava, 2021. 48, 108

[184] K. Cheung and P. Tang, "Sigma-delta modulation neural networks," in *IEEE International Conference on Neural Networks*, 1993, pp. 489–493 vol.1. 48

[185] C. Pehle and J. E. Pedersen, "Norse - A deep learning library for spiking neural networks," Jan. 2021, documentation: https://norse.ai/docs/. [Online]. Available: https://doi.org/10.5281/zenodo.4422025 49

[186] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. 49

[187] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 49

[188] "PySNN." [Online]. Available: https://github.com/BasBuller/PySNN 49

[189] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural computation*, vol. 19, no. 6, pp. 1468–1502, 2007. 49

[190] 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. 49

[191] 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, nov 2015. 50, 79

[192] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7388–7397. 50, 79

[193] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2020. 50

[194] W. Fang, Y. Chen, J. Ding, D. Chen, Z. Yu, H. Zhou, Y. Tian, and other contributors, "Spikingjelly," https://github.com/fangwei123456/spikingjelly, 2020. 50

[195] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: An event-stream dataset for object classification," *Frontiers in Neuroscience*, vol. 11, 2017. 50

[196] A. Vicente-Sola, D. L. Manna, P. Kirkland, G. D. Caterina, and T. Bihl, "Keys to accurate feature extraction using residual spiking neural networks," *Neuromorphic Computing and Engineering*, vol. 2, no. 4, p. 044001, sep 2022. 53, 100

[197] P. Kirkland, D. Manna, A. Vicente, and G. D. Caterina, "Unsupervised spiking instance segmentation on event data using STDP features," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2728–2739, nov 2022. 53

[198] J. Zhang, B. Dong, H. Zhang, J. Ding, F. Heide, B. Yin, and X. Yang, "Spiking transformers for event-based single object tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 8801–8810. 53

[199] A. Safa, L. Keuninckx, G. Gielen, and F. Catthoor, *Sensor Fusion SLAM with Continual STDP Learning.* Springer Nature Switzerland, 2024, pp. 125–153. 53

[200] G. E. P. Box, G. C. Reinsel, G. M. Jenkins, and G. M. Ljung, *Time Series Analysis 5e.* John Wiley & Sons, 2015. 54, 55, 98, 99, 100

[201] C. Deb, F. Zhang, J. Yang, S. E. Lee, and K. W. Shah, "A review on time series forecasting techniques for building energy consumption," *Renewable and Sustainable Energy Reviews*, vol. 74, pp. 902–924, Jul. 2017. 55

[202] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019," *Applied Soft Computing*, vol. 90, p. 106181, May 2020. 54, 55

[203] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Australia: OTexts, 2018. 54, 55, 56

[204] S. Stephen, O. Argwings, and K. Julius, "Application of arima, hybrid arima and artificial neural network models in predicting and forecasting tuberculosis incidences among children in homa bay and turkana counties, kenya," Jul. 2022. 55

[205] S. N. Wood, N. Pya, and B. Säfken, "Smoothing parameter and model selection for general smooth models," *Journal of the American Statistical Association*, vol. 111, no. 516, pp. 1548–1563, Oct. 2016. 56

[206] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5–10, Jan. 2004. 56

[207] R. B. Cleveland, W. S. Cleveland, and I. Terpenning, "Stl: A seasonal-trend decomposition procedure based on loess," *Journal of Official Statistics*, vol. 6, no. 1, p. 3, 03 1990, copyright - Copyright Statistics Sweden (SCB) Mar 1990; Last updated - 2023-11-28. [Online]. Available: https://www.proquest.com/scholarly-journals/stl-seasonal-trend-decomposition-procedure-based/docview/1266805989/se-2 56

[208] Z. Ouyang, P. Ravier, and M. Jabloun, "Stl decomposition of time series can benefit forecasting done by statistical methods but not by machine learning ones," in *The 7th International conference on Time Series and Forecasting*, ser. ITISE 2021. MDPI, Jul. 2021. 56

[209] D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*. Cambridge University Press, 2000. [Online]. Available: http://dx.doi.org/10.1017/CBO9780511841040 56, 99

[210] P. Bloomfield, *Fourier analysis of time series*, 2nd ed., ser. Wiley Series in Probability and Statistics. Nashville, TN: John Wiley & Sons, Jan. 2000. 56, 99

[211] K. Bandara, P. Shi, C. Bergmeir, H. Hewamalage, Q. Tran, and B. Seaman, *Sales Demand Forecast in E-commerce Using a Long Short-Term Memory Neural Network Methodology*. Springer International Publishing, 2019, pp. 462–474. 56, 99

[212] S. Jadid Abdulkadir, H. Alhussian, M. Nazmi, and A. A Elsheikh, "Long short term memory recurrent network for standard and poor's 500 index modelling," *International Journal of Engineering amp; Technology*, vol. 7, no. 4.15, pp. 25–29, Oct. 2018. 57

[213] J. Choi and S. J. Lee, "Consistency index-based sensor fault detection system for nuclear power plant emergency situations using an lstm network," *Sensors*, vol. 20, no. 6, p. 1651, Mar. 2020. 57

[214] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Deep learning with long short-term memory for time series prediction," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 114–119, Jun. 2019. 57, 99

[215] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000. 58

[216] B. Li and T. N. Sainath, "Reducing the computational complexity of two-dimensional lstms," *Interspeech 2017*, 2017. 58

[217] M. Wang, Z. Wang, J. Lu, J. Lin, and Z. Wang, "E-lstm: An efficient hardware architecture for long short-term memory," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 280–291, jun 2019. 99

[218] G. Nan, C. Wang, W. Liu, and F. Lombardi, "Dc-lstm: Deep compressed lstm with low bit-width and structured matrices," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, oct 2020. 58, 99

[219] A. Borovykh, S. Bohté, and C. Oosterlee, "Dilated convolutional neural networks for time series forecasting," *The Journal of Computational Finance*, 2018. 58

[220] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, "Transformers in time series: A survey," 2023. [Online]. Available: https://arxiv.org/abs/2202.07125 58, 99

[221] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf 58

[222] R. Patil and V. Gudivada, "A review of current trends, techniques, and challenges in large language models (llms)," *Applied Sciences*, vol. 14, no. 5, p. 2074, Mar. 2024. [Online]. Available: http://dx.doi.org/10.3390/app14052074 59

[223] S. Ren, B. Tomlinson, R. W. Black, and A. W. Torrance, "Reconciling the contrasting narratives on the environmental impact of large language models," *Scientific Reports*, vol. 14, no. 1, Nov. 2024. [Online]. Available: http://dx.doi.org/10.1038/s41598-024-76682-6 59

[224] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of bloom, a 176b parameter language model," *Journal of Machine Learning Research*, vol. 24, no. 253, pp. 1–15, 2023. [Online]. Available: http://jmlr.org/papers/v24/23-0069.html 59

[225] A. Faiz, L. Jiang, and F. Chen, "Special session: End-to-end carbon footprint assessment and modeling of deep learning," in *2024 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2024, pp. 7–10. 59

[226] K. R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, *Predicting time series with support vector machines*. Springer Berlin Heidelberg, 1997, pp. 999–1004. 59, 99

[227] L. Cao and F. Tay, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1506–1518, 2003. 59

[228] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp. 189–215, Sep. 2020. 59

[229] D. Reid, A. J. Hussain, and H. Tawfik, "Financial time series prediction using spiking neural networks," *PLoS ONE*, vol. 9, no. 8, p. e103656, aug 2014. 60

[230] K. Gao, W. Luk, and S. Weston, "High-frequency trading and financial time-series prediction with spiking neural networks," *Wilmott*, vol. 2021, pp. 18–33, 2021. 60

[231] K. Mateńczuk, A. Kozina, A. Markowska, K. Czerniachowska, K. Kaczmarczyk, P. Golec, M. Hernes, K. Lutosławski, A. Kozierkiewicz, M. Pietranik, A. Rot, and M. Dyvak, "Financial time series forecasting: Comparison of traditional and spiking neural networks," *Procedia Computer Science*, vol. 192, pp. 5023–5029, 2021. 60

[232] H. Fang, A. Shrestha, and Q. Qiu, "Multivariate time series classification using spiking neural networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7. 60

[233] C. Liu, Z. Tao, Z. Luo, and C. Liu, "Mtsa-snn: A multi-modal time series analysis model based on spiking neural network," 2024. 60, 61

[234] V. Sharma and D. Srinivasan, "A spiking neural network based on temporal encoding for electricity price time series forecasting in deregulated markets," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8. 61

[235] S. Y. A. Yarga, J. Rouat, and S. Wood, "Efficient spike encoding algorithms for neuromorphic speech recognition," in *Proceedings of the International Conference on Neuromorphic Systems 2022*, 2022, pp. 1–8. 61

[236] R.-J. Zhu, Q. Zhao, G. Li, and J. K. Eshraghian, "Spikegpt: Generative pre-trained language model with spiking neural networks," 2023. 61

[237] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245 62

[238] M.-I. Stan and O. Rhodes, "Learning long sequences in spiking neural networks," *Scientific Reports*, vol. 14, no. 1, Sep. 2024. [Online]. Available: http://dx.doi.org/10.1038/s41598-024-71678-8 62

[239] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11 106–11 115, may 2021. 62, 102

[240] S. Inam, S. A. Harmain, S. Shafique, M. Afzal, A. Rabail, F. Amin, and M. Waqar, "A brief review of strategies used for emg signal classification," in *2021 International Conference on Artificial Intelligence (ICAI)*. IEEE, Apr. 2021. 64

[241] N. Nazmi, M. Abdul Rahman, S.-I. Yamamoto, S. Ahmad, H. Zamzuri, and S. Mazlan, "A review of classification techniques of emg signals during isotonic and isometric contractions," *Sensors*, vol. 16, no. 8, p. 1304, Aug. 2016. 64

[242] R. Menon, G. Di Caterina, H. Lakany, L. Petropoulakis, B. A. Conway, and J. J. Soraghan, "Study on interaction between temporal and spatial information in classification of emg signals for myoelectric prostheses," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 10, pp. 1832–1842, Oct. 2017. 64, 65, 70

[243] J. Warner, R. Gault, and J. McAllister, "Optimised emg pipeline for gesture classification," in *2022 44th Annual International Conference of the IEEE Engineering in Medicine amp; Biology Society (EMBC)*. IEEE, Jul. 2022. 64, 65, 66

[244] F. Douglas, H. Gover, C. Docherty, G. Shields, K. Leventi, and G. Di Caterina, "An exploration of the optimal feature-classifier combinations for transradial prosthesis control," Jul. 2022, engineering in Medicine and Biology Conference, EMBC ; Conference date: 11-07-2022 Through 15-07-2022. [Online]. Available: https://embc.embs.org/2022/ 65, 66

[245] P. Geethanjali, "Myoelectric control of prosthetic hands: state-of-the-art review," *Medical Devices: Evidence and Research*, vol. Volume 9, pp. 247–255, Jul. 2016. 66

[246] D. C. Toledo-Pérez, J. Rodríguez-Reséndiz, R. A. Gómez-Loenzo, and J. C. Jauregui-Correa, "Support vector machine-based emg signal classification techniques: A review," *Applied Sciences*, vol. 9, no. 20, p. 4402, Oct. 2019. 66

[247] T. Stefanou, D. Guiraud, C. Fattal, C. Azevedo-Coste, and L. Fonseca, "Frequency-domain semg classification using a single sensor," *Sensors*, vol. 22, no. 5, p. 1939, Mar. 2022. [Online]. Available: http://dx.doi.org/10.3390/s22051939 66, 122

[248] D. Xiong, D. Zhang, X. Zhao, and Y. Zhao, "Deep learning for emg-based human-machine interaction: A review," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 3, pp. 512–533, 2021. 66

[249] A. Sultana, F. Ahmed, and M. S. Alam, "A systematic review on surface electromyography-based classification system for identifying hand and finger movements," *Healthcare Analytics*, vol. 3, p. 100126, Nov. 2023. 66, 69

[250] W. Yang, D. Yang, Y. Liu, and H. Liu, "Emg pattern recognition using convolutional neural network with different scale signal/spectra input," *International Journal of Humanoid Robotics*, vol. 16, no. 04, p. 1950013, Aug. 2019. 68

[251] U. Cote Allard, F. Nougarou, C. L. Fall, P. Giguere, C. Gosselin, F. Laviolette, and B. Gosselin, "A convolutional neural network for robotic arm guidance using semg based frequency-features," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2016. 68

[252] S. Shen, X. Wang, F. Mao, L. Sun, and M. Gu, "Movements classification through semg with convolutional vision transformer and stacking ensemble learning," *IEEE Sensors Journal*, vol. 22, no. 13, pp. 13 318–13 325, Jul. 2022. 68, 69, 122, 133, 134

[253] D. Kim, J. Min, and S. H. Ko, "Recent developments and future directions of wearable skin biosignal sensors," *Advanced Sensor Research*, vol. 3, no. 2,

Oct. 2023. [Online]. Available: http://dx.doi.org/10.1002/adsr.20230011869

[254] A. Vitale, E. Donati, R. Germann, and M. Magno, "Neuromorphic edge computing for biomedical applications: Gesture classification using emg signals," *IEEE Sensors Journal*, vol. 22, no. 20, pp. 19 490–19 499, Oct. 2022. 70, 73, 122, 123, 133, 134

[255] M. A. Scrugli, G. Leone, P. Busia, and P. Meloni, *sEMG-Based Gesture Recognition with Spiking Neural Networks on Low-Power FPGA*. Springer Nature Switzerland, 2024, pp. 15–26. 70, 123, 133, 134

[256] B. Hudgins, P. Parker, and R. N. Scott, "A new strategy for multifunction myoelectric control," *IEEE transactions on biomedical engineering*, vol. 40, no. 1, pp. 82–94, 1993. 70, 125

[257] Y. Ma, B. Chen, P. Ren, N. Zheng, G. Indiveri, and E. Donati, "Emg-based gestures classification using a mixed-signal neuromorphic processing system," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 578–587, Dec. 2020. 70, 71

[258] E. Ceolini, C. Frenkel, S. B. Shrestha, G. Taverni, L. Khacef, M. Payvand, and E. Donati, "Hand-gesture recognition based on emg and event-based camera sensor fusion: A benchmark in neuromorphic computing," *Frontiers in Neuroscience*, vol. 14, Aug. 2020. 72, 73

[259] "Thalmic labs," https://github.com/thalmiclabs, 2020. 73, 124

[260] M. Ward and O. Rhodes, "Beyond lif neurons on neuromorphic hardware," *Frontiers in Neuroscience*, vol. 16, Jul. 2022. [Online]. Available: http://dx.doi.org/10.3389/fnins.2022.881598 77

[261] 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. 78, 99

[262] J. Schemmel, S. Billaudelle, P. Dauer, and J. Weis, "Accelerated analog neuromorphic computing," *ArXiv*, vol. abs/2003.11996, 2020.

[263] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[264] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, 2011, pp. 1–4. 78

[265] P. Purohit and R. Manohar, "Field-programmable encoding for address-event representation," *Frontiers in Neuroscience*, vol. 16, Dec. 2022. 79

[266] X. Cheng, Y. Hao, J. Xu, and B. Xu, "Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 1519–1525. 79

[267] P. Kirkland, G. D. Caterina, J. Soraghan, and G. Matich, "SpikeSEG: Spiking segmentation via STDP saliency mapping," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2020. 83, 84

[268] L. R. Iyer and A. Basu, "Unsupervised learning of event-based image recordings using spike-timing-dependent plasticity," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1840–1846. 84

[269] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *ICML*, 2018. 85

[270] W. Gerstner and R. Naud, "How good are neuron models?" *Science*, vol. 326, no. 5951, pp. 379–380, oct 2009. 91

[271] L. R. Iyer, Y. Chua, and H. Li, "Is neuromorphic mnist neuromorphic? analyzing the discriminative power of neuromorphic datasets in the time domain," *Frontiers in neuroscience*, vol. 15, p. 608567, 2021. 98

[272] M. MUTHAMIZHARASAN and R. PONNUSAMY, "A comparative study of crime event forecasting using arima versus lstm model," *Journal of Theoretical and Applied Information Technology*, vol. 102, no. 5, 2024. 99

[273] P. Weidel and S. Sheik, "Wavesense: Efficient temporal convolutions with spiking neural networks for keyword spotting," *arXiv preprint arXiv:2111.01456*, 2021. 100

[274] A. Shaban, S. S. Bezugam, and M. Suri, "An adaptive threshold neuron for recurrent spiking neural networks with nanodevice hardware implementation," *Nature Communications*, vol. 12, no. 1, jul 2021. 100

[275] R. Hogenraad, D. P. McKenzie, and C. Martindale, "The enemy within: Autocorrelation bias in content analysis of narratives," *Computers and the Humanities*, vol. 30, no. 6, pp. 433–439, 1997. 107

[276] Y. Xiao and P. Gong, "Removing spatial autocorrelation in urban scaling analysis," *Cities*, vol. 124, p. 103600, may 2022. 107

[277] A. E. Algüner and H. Ergezer, "Window length insensitive real-time emg hand gesture classification using entropy calculated from globally parsed histograms," *Measurement and Control*, vol. 56, no. 7–8, pp. 1278–1291, Feb. 2023. 122

[278] E. Lashgari and U. Maoz, "Dimensionality reduction for classification of object weight from electromyography," *PLOS ONE*, vol. 16, no. 8, p. e0255926, Aug. 2021. 122

[279] W. Wei, Q. Dai, Y. Wong, Y. Hu, M. Kankanhalli, and W. Geng, "Surface-electromyography-based gesture recognition by multi-view deep learning," *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 10, p. 2964–2973, Oct. 2019. [Online]. Available: http://dx.doi.org/10.1109/TBME.2019.2899222 122

[280] "Lava: A software framework for neuromorphic computing," https://github.com/lava-nc/lava, 2021. 123, 126

[281] L. Chen, J. Fu, Y. Wu, H. Li, and B. Zheng, "Hand gesture recognition using compact cnn via surface electromyography signals," *Sensors*, vol. 20, no. 3, p. 672, Jan. 2020. 125, 133, 134

[282] R. Uwamahoro, K. Sundaraj, and I. D. Subramaniam, "Assessment of muscle activity using electrical stimulation and mechanomyography: a systematic review," *BioMedical Engineering OnLine*, vol. 20, no. 1, Jan. 2021. 125

[283] M. B. I. Reaz, M. S. Hussain, and F. Mohd-Yasin, "Techniques of emg signal analysis: detection, processing, classification and applications," *Biological Procedures Online*, vol. 8, no. 1, pp. 11–35, Dec. 2006. 125

[284] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256. 132

[285] C. R. Carvalho, J. M. Fernández, A. J. del Ama, F. Oliveira Barroso, and J. C. Moreno, "Review of electromyography onset detection methods for real-time control of robotic exoskeletons," *Journal of NeuroEngineering and Rehabilitation*, vol. 20, no. 1, Oct. 2023. [Online]. Available: http://dx.doi.org/10.1186/s12984-023-01268-8 137