eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor Thesis

Degree on Artificial Intelligence

Toward Biologically Plausible Representations of Neural Activity in Artificial Intelligence

Izei Múgica Martínez

Advisors
Jon Vadillo Jueguen
Roberto Santana Hermida

June 21, 2025

Acknowledgements

Many days have passed since I first dreamed of living in peace, embracing my true self, surrounded by all the cherished souls I have encountered over the years. This work stands as both the closing of one chapter and the dawn of another, yet I know that each of you who walked this journey with me will forever live within these words.

With this in mind, I dedicate this work to all of you: all the teachers who ignited my passion; Jon and Roberto, because your guidance illuminated this beautiful path; Usue, Borja and Naiara, because you believed in me and opened the doors to the world of research; Josu, because you taught me the meaning of perseverance and effort; and Olatz, Gorka and Oier, because your passion remains an inspiration.

Of course, I could not forget about you, my family and friends. I also dedicate this work to all of you: aita, ama and Aitor, because your wisdom and care are my compass; Lorea López, because you show me the warmth of kindness and joy; Ander Larruskain and Luken Munguira, because you are my steadfast and gorgeous supports; Maddi Asenjo, because your active listening and kindness are both marvelous; Nahia Sarasa, Lucía Arzallus and Iker Bilbao, because you have been the most precious additions to my life in this past year; Ibai Murillo, because after sharing eight days on a motor home with you on our trip to Germany, I feel like I can endure anything now; Gorka Miranda, because your tireless rehearsals carry the spirit of renewal; Iker Vidal, because your support comes in the form of kind and knowing smiles; you, mysterious person, because you are the one with whom I dream in the silence of the night; and you, amama, because your embrace reaches me every night from beyond the stars.

Each of you lives on in my heart, woven into the fabric of this work, forever a part of my journey.

- COLLINA PRESSO NAGASAKI -

Casa giapponese, terrazzo e giardino.
In fondo, al basso, la rada, il porto,
la città di Nagasaki.

Abstract

In recent years, artificial intelligence, particularly machine learning, has achieved remarkable successes across diverse domains by employing layers of simple perceptron-style artificial neurons. However, these models abstract away many key biophysical phenomena that govern real neuronal dynamics, such as variable spike thresholds, channel-density effects and saturation, raising the question of whether greater biological fidelity might yield new computational advantages.

This work aims to bridge that gap by exploring how insights from computational neuroscience can be applied in the design of neural units for machine learning. In order to do so, extensive and critical research is conducted into existing neurocomputational and neural-machine-learning models, analysing both their strengths and shortcomings. Building on that analysing and on foundational neurocomputational models, we formulate a single-neuron architecture that treats membrane potential as an explicit dynamical variable, governed by the quadratic integrate-and-fire model. We derive closed-form expressions for the analytical gradient and show how the adjoint-state method enables efficient computation.

In order to validate the theoretical development, we conduct a series of experiments that test all those theoretical components by training our neural model to solve the XOR problem and the concentric circumferences problem for a specific pair of radii. These are linearly nonseparable problems; they can not be solved by classic logistic regressors without requiring feature expansion or additional neurons to build a neural network. In our experiments, the neural model achieves perfect separation (100% accuracy) on both the XOR problem and on the concentric circumferences problem. These results suggest that embedding neurocomputational principles into machine learning models can enrich expressivity beyond traditional static perceptrons, although with trade-offs in numerical integration and training efficiency. In conclusion, this thesis demonstrates that "neurons as dynamical systems" is not merely a theoretical viewpoint but a practical design paradigm.

On Sustainable Development Goals

The Sustainable Development Goals (SDGs), adopted by the United Nations in 2015, provide a global scheme for ending poverty, protecting the planet and ensuring prosperity for all by 2030. This initiative, which considers seventeen interlinked goals, addresses the root causes of imbalance and determines shared responsibilities for governments, businesses and researchers. Among these, the ninth sustainable development goal, commonly referred to as "Industry, Innovation and Infrastructure" and which targets the creation of resilient infrastructure, the promotion of inclusive and sustainable industrialisation, and the conception of innovation ecosystems, has been a key participant during the development of this thesis. In fact, the fourth target of this goal urges the upgrading of infrastructure and the retrofit of industries to make them sustainable, with increased resource-use efficiency and greater adoption of clean and environmentally sound technologies. In addition, target 9.b calls for expanded research and development and greater technological capacity in all countries.

This thesis on biologically plausible neural models speaks directly to these targets. Conventional artificial intelligence frameworks rely on dense, resource-exhaustive computations that demand large data centers and heavy hardware. In contrast, the spiking-neural-ODE formulation this thesis has developed attempts to mimic the sparse, event-driven messaging of the brain, which is known to be remarkably efficient in terms of electric consumption. As such, apart from addressing theoretical bounds and limitations of existing neural models, it tries to shed light on an efficient framework for developing artificial intelligence. The reformulation of the artificial neuron that has been developed in this thesis embodies the fourth target of the ninth sustainable development goal: it offers a pathway toward artificial intelligence infrastructures that consume far less energy, making edge devices and remote deployments both feasible and sustainable. On the innovation front, translating neurobiological insights and signals into mathematical models establishes a baseline for neuromorphic hardware chips designed to process information like neural tissue. This interdisciplinary fusion exemplifies target 9.b, as it expands research horizons across computational neuroscience, electrical engineering and possibly materials science.

Beyond technical metrics, the broader impacts are considerable: energy-efficient neural systems can power self-optimising manufacturing lines, environmental sensors, etc., all with minimal carbon footprint. In reshaping the infrastructure of artificial intelligence to be lighter, greener and more adaptable, this research considers not only theoretical frontiers, but also tangible and equitable progress toward the vision of Sustainable Development Goals.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

The favorable reception that artificial intelligence has had in society in recent years has highlighted its capabilities, and many of us will agree that it has made our daily lives easier. Of course, this power and these capabilities are the result of many years of extensive research, which continues today with numerous and diverse lines of investigation, showing signs that artificial intelligence will have a prominent presence in the near future. As such, behind this surge driven by exhaustive research, there are numerous fields, characteristics, proposals and discoveries, such as data collection and processing techniques, advanced artificial intelligence techniques and architectural proposals for building models. It is this last field, perhaps, that has sparked the greatest interest in recent years, thanks to the potential that neural-based machine learning and deep learning have demonstrated in solving highly complex machine learning tasks, tasks that, until not long ago, were considered long-term challenges.

But what are those two fields and how have they managed to establish themselves as the main characters in building artificial intelligence models? In simple terms, neural-based machine learning and deep learning are subfields of machine learning based on the concept of the artificial neuron as the foundation for solving various tasks in this field. Although there is no formal distinction between where neural-based machine learning ends and where deep learning begins, it is generally agreed that neural networks of various hidden layers are the building blocks of deep learning, whereas neural-based machine learning encompasses a wide array of configurations over artificial neurons. While this concept may seem complex, and it is true that cutting-edge deep learning models integrate a wide range of sophisticated techniques built upon this foundation, the underlying theoretical and mathematical principles are not as complicated as they might seem. Essentially, an artificial neuron is a function that takes an input data point, computes a linear combination of its numerical values, and then typically applies a nonlinear function σ to the result. This process produces an output that elegantly captures and propagates the interactions between the input data variables, as illustrated in Figure 1.1.

This model, built on Frank Rosenblatt's perceptron [1] (which, at the same time, can

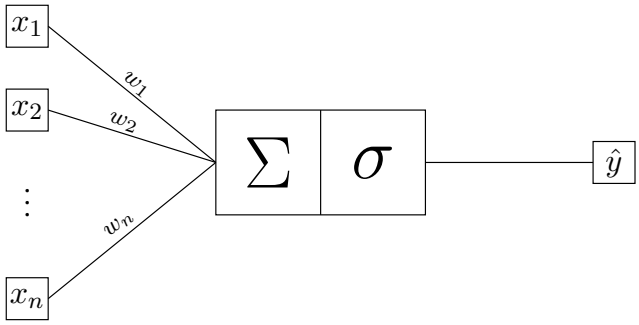

Figure 1.1: Diagram of the flow of information in an artificial neural network. Input data is combined via a linear combination $w_1x_1 + w_2x_2 + \cdots + w_nx_n$, after which the σ function is applied to the linear combination, resulting in a value $\hat{y}$.

be considered an interpretation of the neural model introduced by McCulloch and Pitts in 1943 [2]), represents an attempt to model the functioning mechanism of neurons. Neurons constantly modulate the electrical flow that passes through them, and depending on the characteristics of this flow and the neuron's physiology, they produce a specific output. In any case, does this model yield good results when constructing models, whether simple or complex, based on this architecture? Furthermore, can we say that the model accurately represents, from both a biological and functional perspective, the mechanisms by which neurons operate? The reason behind these questions is that if the key to developing artificial intelligence models with great capabilities lies in integrating or simulating the neuronal mechanisms of living beings, then a biologically plausible neural model could be a reasonable research objective.

That said, it is well known that the simple perceptron has laid the foundation for milestones such as diffusion models and language models based on the Transformer architecture [3]. This suggests that there is sufficient evidence to support the robustness that the simple perceptron offers, especially when multiple artificial neurons are combined to form artificial neural networks. However, in terms of biological plausibility, we cannot claim that the model accurately reflects the functioning of neurons, as it overlooks several key properties in the propagation and processing of the electrical flow. These include the feedback signal resulting from historical spikes, the variable response to constant electrical currents, saturation in response to certain impulses and electrical flows, the density of ion channels, the effects that neurons have on their environment and the effects that the environment has on neurons.

Under these circumstances, one could hypothesise that if a model as seemingly "simple" as the perceptron has served as the foundation for groundbreaking models like GPT-3 [4] or Stable Diffusion [5], a more complex model or one that more accurately reflects the functioning of neurons could represent a significant advancement in the field of machine learning. Moreover, it is well known that the functioning of brains, composed of billions of neurons, is considerably more efficient than that of today's most cutting-edge models. Therefore, exploring new possibilities could also lead to the development of efficient models.

It is within this context that this Bachelor Thesis emerges. The main objectives are to analyse the progress made in computational neuroscience to understand how neurons function

and to employ some of its key ideas to propose and develop a biologically plausible neural model that could serve as a foundation for developing a potential neural architecture. We acknowledge that the combined effect of interconnected neurons may be responsible for their extraordinary capabilities. However, this thesis will focus on developing the foundations for a computational neural model without addressing the synchronisation of neurons.

1.1 Methodological Description of Objectives

We have devised a series of objectives that encompass what a research project of these characteristics should contain, developing this thesis in accordance with them:

1. **Objective 1: Literature Review of Computational Neuroscience.** This objective seeks to introduce us to the field of computational neuroscience. Analysing the anatomical, electrophysiological and dynamical properties of neurons and how they relate to computational components is essential to obtain a critical viewpoint of how artificial intelligence is currently being developed, with a particular interest in machine learning models and how they tie with other neurocomputational models. Moreover, building a biologically plausible neural model should also be the result of a critical review of this field, since it allows us to understand the decisions taken when building existing biologically plausible models, as well as to analyse their strengths and limitations.

2. **Objective 2: Development of a Biologically Plausible Neural Model.** This is the most relevant objective in the thesis, as it requires understanding and implementing the neurocomputational properties or components that could enhance the biological plausibility of state-of-the-art artificial intelligence models. Implementation should be coherent and robust, supported by a rigorous mathematical formulation and a derivation of results. As such, this objective asks for the development of a biologically plausible neural model by incorporating the principal components identified during the literature review, as well as for the development of the mathematical formulation for the model and the derivation of a complete training procedure.

3. **Objective 3: Experimental Validation and Drawing Conclusions.** Upon formalising a theoretical framework for a biologically plausible neural model, it is mandatory to conduct some experiments in order to assess and validate the correctness of the theoretical results, in addition to the performance that the model exhibits (especially in comparison with other machine learning models). Therefore, this requires an implementation of the model. Drawing conclusions should be a natural end to the project.

1.2 Planification

This thesis has been developed within a time frame of four months, from mid-February to mid-June, distributing time to develop each principal section in accordance with the theoretical workload of each section. Figure 1.2 explicitly shows the planification that has been suggested

to develop the project, for a total of 325 hours. In addition, Figure 1.3 shows the Gantt chart that contains the scheduling for this project together with the tasks that have been identified to achieve each defined objective. We consider those tasks to be the following:

1. **Task 1: Research into anatomical and electrophysiological characteristics of neurons.** Before delving into their neurocomputational properties, it is essential to understand what neurons are, what their components are and how electricity is transmitted through them by understanding electrochemical gradients and their implications. This task corresponds to the completion of Objective 1.

2. **Task 2: Study neurons as dynamical systems.** One of the fundamental aspects relating neuroscience and computation is that neurons can be characterised through dynamical systems. Their behaviour and the phenomena they exhibit correspond to internal states that evolve over time. As a consequence, it is essential to study how dynamical systems characterise neurons in order to acquire essential knowledge about neural modelling. This task corresponds to the completion of Objective 1.

    a) **Task 2.1: Study one-dimensional systems.** These are the most simple dynamical systems that should allow us to understand the most basic concepts behind neural behaviour and how neurons are characterised through this field.

    b) **Task 2.2: Study two-dimensional systems.** These are some more complex dynamical systems that explain other phenomena that can not be explained via one-dimensional systems. Acquiring a critical viewpoint requires studying what two-dimensional systems can offer, so that modelling decisions are made on a robust foundation.

3. **Task 3: Study existing neural models.** Before working on a proposal, it is essential to study which modelling attempts have been made to emulate neural behaviour and to employ neurons as resources to solve machine learning tasks. This requires both delving into classical neurocomputational models and into state-of-the-art machine learning models. This task corresponds to the completion of Objective 1.

4. **Task 4: Analyse which components could be integrated to build a biologically plausible neural model.** The first step toward building a biologically plausible neural model consists of studying which relevant components could enhance the biological plausibility of existing artificial intelligence models. This task corresponds to the completion of Objective 2.

5. **Task 5: Formalise all the components and the architecture of the neural model.** Once all the aspects to integrate have been identified, those must be included to build a neural model on top of current existing neural machine learning models. This requires defining how data should be represented, how information should flow, which components will dictate whether the neuron is performing as expected and how the neuron can be trained. Prioritising existing learning schemes is important. This task corresponds to the completion of Objective 2.

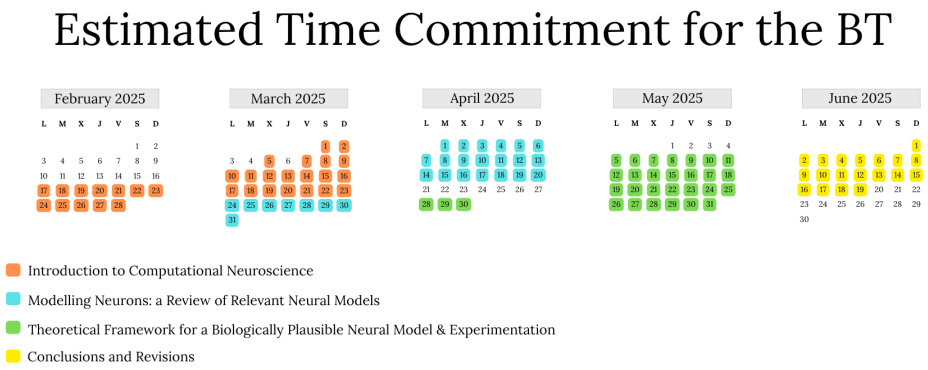

Figure 1.2: Planification guideline for the project. Shown is the distribution of days allocated to each of the four principal blocks of this thesis.

a) **Task 5.1: Derive an efficient training procedure.** Once a general-purpose training framework has been developed, efficiency should be addressed, analysing different alternatives or solutions that could enhance the efficiency of the training process.

6. **Task 6: Implement the whole developed framework to validate it later through experiments.** Once the theoretical development has been conducted, evaluating and validating the development is essential. However, that requires implementing the model beforehand. This task corresponds to the completion of Objective 3.

7. **Task 7: Devise a methodology for the experimentation needed to validate the theoretical development of the neural model.** Validating several aspects that we want to highlight requires designing suitable and descriptive experiments. In fact, the biological coherence of the model and the resolution of problems that classic neural machine learning models can not solve should be considered. This task corresponds to the completion of Objective 3.

8. **Task 8: Conduct experiments and draw conclusions.** Once the methodology has been established, the proposed experiments should be conducted, providing a critical reflection on the results obtained and drawing meaningful conclusions on the entire project. This task corresponds to the completion of Objective 3.

We must highlight the relevant amount of time invested into research, as delving into computational neuroscience is a challenging task that requires both profound mathematical, anatomical and electrophysiological expertise. Nevertheless, we consider this has been essential in order to develop a critical viewpoint to address the strengths and shortcomings of existing neural models, especially those employed in machine learning, as well as to include several neurocomputational components and build a biologically plausible neural model within a robust basis. In addition, developing a neural model has also required meticulously analysing existing attempts to offer alternatives that try to incorporate these ideas to improve

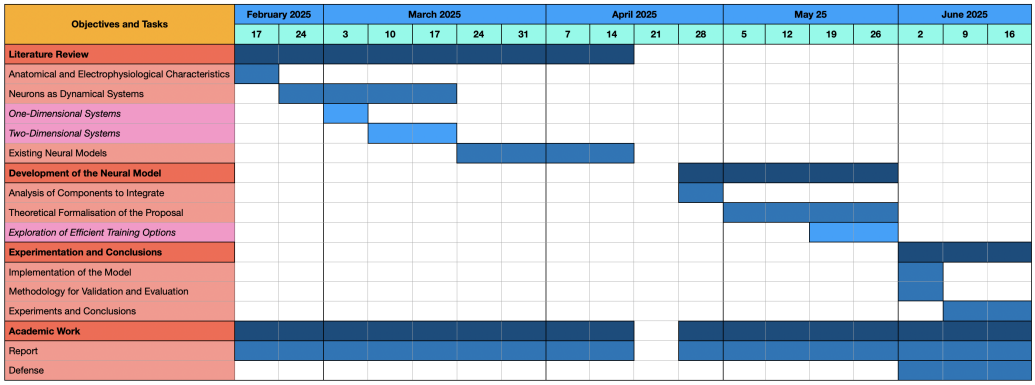

Figure 1.3: Gantt diagram that details time scheduling. Shown is the distribution of time allocated to the completion of each task and objective.

biological plausibility. In summary, this work has required extensive research in a complex field such as computational neuroscience.

1.3 Structure of the Report

As for the structure of this work, this report contains four main sections that address the completion of the aforementioned objectives:

1. Chapters 2 and 3 provide a broad introduction to computational neuroscience, presenting the most important biological and electrophysiological concepts related to neurons while considering computational aspects in parallel. The most significant neural models in the literature are reviewed, as well as how neural-based machine learning models try to mimic these physical phenomena.

2. Chapter 4 builds on the ideas introduced in the previous two chapters, constructing and describing the theoretical framework for a potential functional neural model, with a particular focus on its feasibility and biological plausibility. We find it very interesting to incorporate the differential equations that characterise the behaviour of some neurons into existing machine learning frameworks, providing the analytical gradient of the loss with respect to the learnable parameters as well as an alternative technique to efficiently compute that gradient.

3. Chapter 5 covers the implementation and the experiments conducted to evaluate and validate the capabilities of the proposed model. Aside from studying whether the model exhibits coherent performance during training, its capacity to solve nonlinear problems is analysed and compared against other implementations of logistic regressors.

4. Finally, Chapter 6 summarises the entire study, providing an in-depth discussion of the conclusions drawn and outlining future research directions that may arise from this work.

CHAPTER 2

Introduction to Computational Neuroscience

Before exploring biologically plausible model proposals, it is essential to understand the biochemical mechanisms that govern neuronal behaviour. These mechanisms serve as the foundation for determining whether a neural model aligns with biochemical parameters and assessing the extent to which it exhibits plausible behaviour. One could argue that understanding how neurons function is an extremely challenging task, as it requires deep knowledge of biology, chemistry, physics and mathematics. However, fortunately, computational neuroscience allows us to reduce these complex constraints into mathematical models (varying in complexity as needed), effectively explaining neuronal behaviour from its computational perspective [6]. This does not mean that biological, physical and other perspectives should be disregarded, but rather that all physical phenomena can be explained by analysing the computational properties of neurons, something we are interested in, as those models will be the basis for transitioning from physical models to deep learning models, for instance.

As such, this section aims to provide general knowledge on neuronal anatomy, electrophysiology and excitability, as well as key concepts from a computational perspective. The goal is both to understand how neurons function and to justify that the computational approach explains the most significant phenomena they undergo. One of the key properties of neural models is that they are dynamical systems, that is, systems that evolve over time. Consequently, a significant portion of this section will focus on dynamical systems and their capacity to model neuronal behaviour, with particular emphasis on the well-known *Hodgkin-Huxley* model. However, we will also discuss how neural modelling is approached using simplified yet powerful one- and two-dimensional systems. All the concepts introduced in this chapter will be fundamental when presenting classic neural models in Chapter 3 and when discussing how current deep learning models try to mimic neuronal behaviour, also in Chapter 3. Naturally, these concepts will also be essential to justify a neural model proposal in Chapter 4.

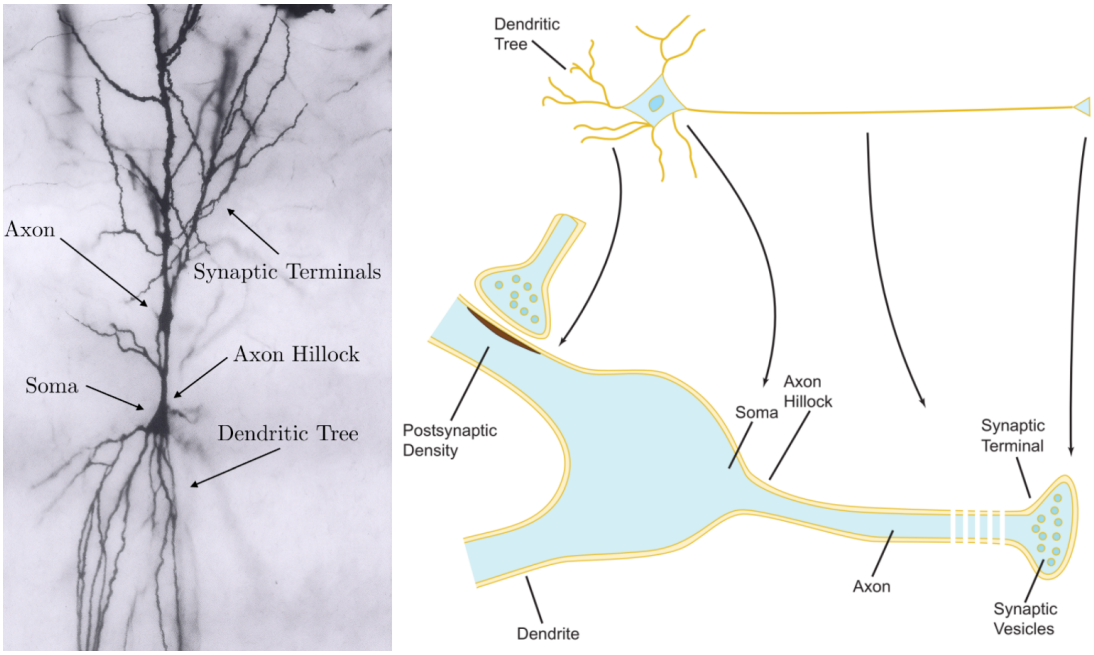

Figure 2.1: A single Golgi-stained neuron in the hippocampus (left) and the ultrastructure of the neuron (right). Images adapted from [7].

2.1 The Neuron

The neuron is a type of eukaryotic cell and the main component of the nervous system. It is well known to be an excitable cell that propagates electrical signals throughout the nervous system. These electrical signals are responsible for movement, regulation, cognition and other vital functions, making neurons remarkably complex cells. In addition, since neurons play different roles, they are typically distributed throughout the body, via the nervous system, based on their structure and function. In such a scenario, the reference to the nervous system inherently involves the brain, as it is considered the most important organ, serving as the central control in all vertebrates and most invertebrates. Moreover, the brain can arguably be considered the most complex structure produced by evolution, further emphasising the intricate nature of neurons. It is therefore important to understand the components of a neuron and how it propagates electrical signals. This section will focus on the anatomical perspective, leaving electrophysiology for Section 2.2.

2.1.1 Anatomy and Histology

This subsection aims to provide an overview of the structure of neurons. Although there is a wide variety of shapes and structures among different types of neurons, most share common anatomical features that will be discussed here. Figure 2.1 includes two images that illustrate this concept, one showing a real hippocampal neuron (left) and the other a diagram that highlights its main structures (right). The two images align well, as the structures depicted in

the diagram on the right can be easily located on the left image, as shown with the arrows and labels. As such, the main components that need to be addressed here are the **soma**, the **dendritic tree** or the **dendrites**, the **axon** (together with the **axon hillock**) and the **synaptic terminals** (also known as **axon terminals**).

2.1.1.1 The Soma

The soma is the core component of the neuron, as it contains the most important organelle found in a cell: the nucleus, which houses the neuron's DNA and controls gene expression; the rough endoplasmic reticulum and ribosomes, which are involved in protein synthesis; the Golgi apparatus, which modifies and packages proteins; mitochondria, which provide energy for cellular functions; and lysosomes and peroxisomes, which help with waste breakdown and cellular maintenance. Therefore, the soma is a crucial component of the neuron, as it is responsible for most protein synthesis and houses the genetic material of the cell [8].

2.1.1.2 The Dendritic Tree (Dendrites)

Dendrites are highly branched structures that typically extend from the cell body and, in some cases, from the proximal regions of the axon, collectively forming what is known as the dendritic tree. Dendrites serve as primary input carriers, since they contain membrane protrusions called dendritic spines, which receive neuronal information at synapses and carry it through the neuron. This is made possible by a set of proteins in the plasma membrane, which specialise the membrane to allow dendrites to receive and integrate information from other nerve cells [7].

Synapses are among the most fascinating structures involved in signal propagation between neurons, as their main function is to transmit electrical or chemical signals from one neuron to another [9]. In electrical synapses, neurons are connected through gap junctions (membrane channels that allow the direct exchange of cytoplasmic substances), allowing the propagation of electrical signals [10]. In chemical synapses, communication occurs through neurotransmitters: the presynaptic neuron releases specific neurotransmitters that bind to receptors on the dendritic spines of the postsynaptic neuron, triggering an electrical or chemical response.

However, it is important to note that dendrites do not function solely in information reception. Some dendrites are capable of transmitting electrical signals, allowing the input/output process to occur entirely within the dendritic tree [7].

2.1.1.3 The Axon

The axon is a thin tube-like structure that extends from the cell body through a cone-shaped thickening known as the axon hillock. It is primarily responsible for transmitting nerve signals away from the soma, as well as carrying information back to it. It should be noted that the axon can vary in length from a few micrometers to several meters, typically branching to form synaptic terminals, making it a key player in signal transmission [7]. This is why electrophysiological properties will primarily be analysed with a focus on axons, as some

are large enough to be studied and have fewer interneuron connections. The behaviour of dendrites and synapses should particularly be considered when working on synchronisation, such as when working on neural networks, although this is out of the scope of this thesis.

2.1.1.4 Synaptic Terminals (Axon Terminals)

Synaptic terminals are structures located at the end of the axon, furthest from the soma, and they represent the other main component that contains synapses that connect neurons. While dendrites receive input from other neurons, axon terminals transmit electrical or chemical signals to postsynaptic neurons, completing the cycle that explains the flow of information between neurons [11].

2.1.2 The Membrane and Electrochemical Gradients

After explaining the core components of a typical neuron, it is important to address its membrane, as much of a neuron's behaviour is governed by its membrane. Computational neuroscience builds on the membrane's role as a capacitor and how it regulates the flow or blockage of electrical currents. Therefore, understanding the membrane is key to understanding how electrical signals propagate through neurons, how action potentials are generated and the mathematical equations that govern neuronal behaviour. This subsection will first cover the main biochemical properties of the membrane, before delving into the electrophysiological implications of this structure.

The neuronal membrane is a lipid bilayer that encloses the cell and contains numerous embedded protein structures [12]. As a lipid bilayer, the membrane acts as an electrical insulator, preventing direct flow of electrical currents through it. However, many of those embedded protein structures are electrically active (these structures are known as channels), thus allowing ions to pass from one side of the membrane to the other one. This combination allows the membrane to act as a capacitor, because ions accumulate on either side of the membrane, and it is only through the channels that those ions can flow, altering the membrane voltage, i.e., the difference in electrical potential between the inner and outer cell. This is an essential property of the membrane, as it accounts for most of the phenomena in which the neuron is involved, but it opens many questions that need to be addressed.

First, how do the electrically active embedded protein structures allow ions to flow through them? It is not as simple as for them to be openings in the membrane, because they have distinct characteristics that affect the flow of ions. The membrane contains an array of channels that can be classified into **gated** and **non-gated**. Non-gated channels, also known as **leak channels**, are among the simplest, as they can be understood as the aforementioned openings in the membrane that allow ions to flow according to their electrochemical gradient. As such, these channels lack any internal mechanisms that need to be explained to understand ion flow; it all comes down to the electrochemical gradient, which will be explained below. However, this does not mean that any ion can flow through leak channels, as they can be selective for certain ions. Variations in size and shape allow only specific ions to pass through, effectively filtering out undesired ions. There are even membrane-impermeable ions that are

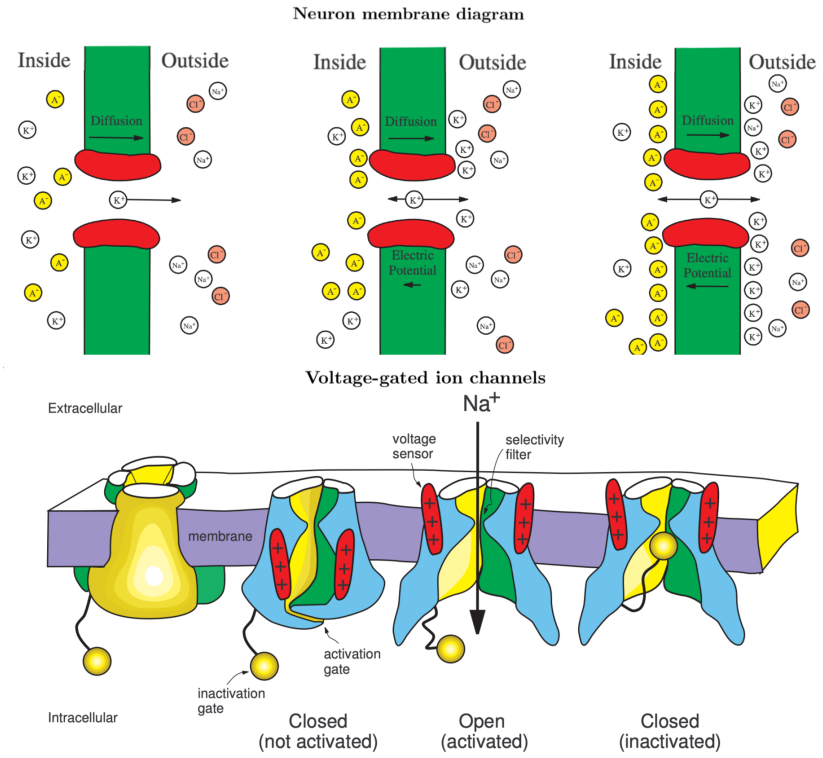

Figure 2.2: Diagrams illustrating the core components of the neuron membrane. The top figure shows the typical ion flow through membrane channels, while the bottom figure illustrates the functioning of voltage-gated channels. Images taken from [6].

unable to traverse the cell membrane (these are typically negatively charged ions). Figure 2.2 (top) shows a diagram that illustrates the flow of ions through leak channels.

The case for gated channels is quite different, as they can not only be selective, but also open, closed, activated and inactivated depending on factors such as membrane voltage. Therefore, certain ionic currents will pass through these channels only under specific conditions and always in accordance with the electrochemical gradient. With that in mind, the following states can be distinguished for gated channels, summarised in Figure 2.2 (bottom):

- **Open state.** A gated channel is said to be open when it is both structurally open and not inactivated. In this state, gated channels resemble leak channels in that they allow ionic currents to flow through them, but it is important to remember that they are still gated channels regulated by specific mechanisms. As gated open channels need to be structurally open and not inactivated, this state can be achieved either through structural modifications of the proteins that allow the passage of ions or by deinactivating an already open channel (that is, by opening the inactivation gate, which in this case is the sole barrier). Conversely, if the channel is structurally closed or the inactivation gate is blocking the passage, the channel is considered closed.

- **Closed state by no activation.** A gated channel is said to be closed due to lack of activation when its closure is purely morphological and dependent on the activation gate. For instance, if the membrane voltage is not high enough, some voltage-gated channels may not open their activation gate so that ions can pass through them.
- **Closed state by inactivation.** A gated channel is said to be closed due to inactivation when the activation gate is open but the inactivation gate is closed. In this state, ions could theoretically pass through the channel, as the structure permits it, but a blockage caused by the closed inactivation gate prevents ion flow. This means that the channel is structurally open yet functionally closed. This distinction is crucial for understanding the generation of action potentials in neurons.

To conclude the anatomical discussion of the membrane, it is worth noting that not all gated channels possess inactivation gates. Moreover, their opening and closing changes over time in response to intrinsic properties such as shifts in membrane voltage.

As for the flow of ions through open channels, motion and flow are described by their electrochemical gradients. It is essential to understand that the inner and outer cell contain different concentrations of ions as well as different electric potentials, and this causes ions to move in order to balance concentration and electrical imbalances. On the one hand, **diffusion**, induced by temperature and the **concentration gradient**, makes the ions flow from high-concentration regions to low-concentration regions until they become evenly distributed. This is a physical process subject to Fick's second law [13], which states that given $\phi(x,t)$, the function that describes the concentration of a given ion in a location x and on a time t (given in mol/m^3), concentration evolves by the Equation (2.1):

$$\frac{\partial \phi}{\partial t}(x,t) = D\frac{\partial^2 \phi}{\partial x^2}(x,t), \tag{2.1}$$

where $D = D_0 \exp\left(-\frac{E_A}{RT}\right)$, D_0 being the maximal diffusion coefficient (given in m^2/s); E_A the activation energy for diffusion (given in J/mol); T the absolute temperature (given in K); and $R \approx 8.31446$ J/(mol $\cdot$ K) the universal gas constant. On the other hand, the **electrical force** drives the ions toward regions with opposite charge. In this case, this process is subject to Coulomb's law, which states that the electrical force between two charged particles is proportional to their charges and inversely proportional to the distance between them, as Equation (2.2) shows:

$$|F| = k_e \frac{|q_1||q_2|}{r^2}, \tag{2.2}$$

where $|F|$ is the magnitude of the electrical force (given in N); $k_e = \frac{1}{4\pi\epsilon_0}$ is Coulomb's constant (given in N $\cdot$ m^2 $\cdot$ C^{-2}); $|q_1|$ and $|q_2|$ are the charge magnitudes of the two particles, respectively (given in C); and r is the distance between both particles.

Interestingly, these two physical processes can drive ions in opposite directions. For example, consider K^+, an ion whose concentration is much higher inside the cell than outside (approximately 140 mM vs. 5 mM). Leak channels are particularly selective for potassium ions

and, as a result, the concentration gradient drives K^+ ions out of the cell through these channels. However, as more potassium ions exit, the inside of the cell becomes increasingly negative, strengthening the electrical force that pulls K^+ ions back in. Eventually, these opposing forces can reach a balance, leading the membrane to settle at its resting potential. Ultimately, ion flow across the membrane is governed by both the concentration and electrical gradients. Together with the dynamic behaviour of gated channels over time, these factors determine how electrical signals propagate through the neuron, and this is key to understanding how neurons work. Figure 2.2 (top) presents an intuitive example of how electrochemical gradients determine ionic flow through the membrane.

2.2 Electrophysiology of the Neuron

This subsection aims to explain the fundamental concepts of neuronal electrophysiology that are necessary to motivate all the decisions that are made (and therefore the decisions that will be taken in this thesis) when developing computational models for neurons. Much of the content that will be presented aims to show what the theoretical background is for different neuronal phenomena, bridging the gap between biochemical and computational aspects.

2.2.1 Nernst Potential and Goldman Equation

One of the most important mathematical concepts in neuronal electrophysiology is related to the balance of the forces that drive ion species through the membrane channels. We have recently shown that electrochemical gradients are the source of ionic transmembrane flows, but we must delve into this from a theoretical perspective. Both gradients, which are responsible for flow under charge and concentration imbalances, produce accumulation of charges on either side of the membrane, thus resulting in a difference of electric potential between the inner and outer cell (this has a direct effect on the electrochemical gradients, which further alter the electric potentials, entering into a cyclic process). This electric potential difference is known as the **membrane voltage**, and it is responsible for altering the electrochemical gradients cyclically; in other words, the membrane voltage is responsible for the vast majority of the phenomena that occur within neurons. As such, many concepts that will be addressed in this thesis will be strongly related to membrane voltage.

Since electrochemical gradients, which naturally push the neuronal system to an equilibrium such that they all cancel each other out and flow is stopped, result in complex dynamics of ions, and considering voltage is what determines how electrochemical gradients will evolve over time, a natural question one may ask is whether there exists some value for the membrane voltage that cancels out those gradients. Indeed, such a value, known as the **Nernst potential**, does exist, but it only explains how electrochemical gradients cancel out for a given ionic species, without accounting for the fact that the membrane may be actually permeable to various ions (in any case, it is also interesting to know the behaviour exhibited by the membrane for a certain ionic species). In such a scenario, the value of the equilibrium potential or the Nernst potential for the membrane for a given ionic species can be calculated by the Nernst

equation [14], as Equation (2.3) shows:

$$E_{\text{ion}} = \frac{RT}{zF} \ln\left(\frac{[\text{Ion}]_{\text{out}}}{[\text{Ion}]_{\text{in}}}\right), \tag{2.3}$$

where $[\text{Ion}]_{\text{in}}$ and $[\text{Ion}]_{\text{out}}$ are concentrations of the ions inside and outside the cell (given in M), respectively; R is the universal gas constant; T the absolute temperature; $F \approx 96,485$ is Faraday's constant (given in C/mol); and z the valence of the ion.

However, considering that the membrane is permeable to many ions, such as potassium, calcium, sodium and chloride, it is clear that we need to account for the contributions of each ion to the electrochemical gradients in order to calculate the resting or equilibrium potential of the membrane. In such a scenario, these contributions are summarised via the Goldman equation, which states that the resting potential when considering monovalent ions is determined by the Equation (2.4):

$$E_{\text{m}} = \frac{RT}{F} \ln\left(\frac{\sum_{\text{ion}^+\in\text{Ions}} P_{\text{ion}^+}[\text{ion}^+]_{\text{out}} + \sum_{\text{ion}^-\in\text{Ions}} P_{\text{ion}^-}[\text{ion}^-]_{\text{in}}}{\sum_{\text{ion}^+\in\text{Ions}} P_{\text{ion}^+}[\text{ion}^+]_{\text{in}} + \sum_{\text{ion}^-\in\text{Ions}} P_{\text{ion}^-}[\text{ion}^-]_{\text{out}}}\right), \tag{2.4}$$

where P_{ion} refers to the selectivity for that ion by the membrane (given in m/s). It turns out that considering the most abundant and monovalent ions in the neuron (potassium, sodium and chloride), together with the typical selectivity values of $P_{\text{K}^+} = 1$, $P_{\text{Na}^+} = 0.04$, $P_{\text{Cl}^-} = 0.45$ [15], we have that the resting potential of the membrane is approximately

$$\begin{aligned} E_{\text{m}} &\approx \frac{8.314 \times 310}{96480} \times \ln\left(\frac{P_{\text{K}^+}[\text{K}^+]_{\text{out}} + P_{\text{Na}^+}[\text{Na}^+]_{\text{out}} + P_{\text{Cl}^-} \times [\text{Cl}^-]_{\text{in}}}{P_{\text{K}^+}[\text{K}^+]_{\text{in}} + P_{\text{Na}^+}[\text{Na}^+]_{\text{in}} + P_{\text{Cl}^-} \times [\text{Cl}^-]_{\text{out}}}\right) \\ &= \frac{8.314 \times 310}{96480} \times \ln\left(\frac{1 \times 5 + 0.04 \times 145 + 0.45 \times 4}{1 \times 140 + 0.04 \times 10 + 0.45 \times 110}\right) \approx -0.072 \text{ V} = -72 \text{ mV}. \end{aligned}$$

We shall refer to the resting membrane potential as V_{rest}, since in electrophysiology E_{m} represents the membrane potential, whether it is the resting one or not.

This formula accurately describes the typical resting membrane potential commonly cited in the neuroscience literature. Understanding both Nernst potentials and the Goldman equation is crucial, because it gives invaluable information on a neuron's behaviour at rest.

2.2.2 Ionic Currents and the Membrane as a Capacitor

Since the membrane voltage alters the electrochemical gradients that cause the flow of ions across the membrane, it follows that the membrane is responsible for the ionic currents that are generated, currents that contain the electrical signals that are responsible for the most complex phenomena evolution has brought about, such as cognition. Therefore, it is vital to

understand how such currents are generated and their relationship with membrane voltage. All of this will be addressed in this subsection.

As far as ionic currents are concerned, Nernst equations tell us that for a given ionic species and equilibrium potential E_{ion}, the electrochemical gradients cancel each other out, thus resulting in no net movement of charges across the membrane for that particular ionic species. However, for every other voltage value (even if the membrane is at its rest potential), electrochemical gradients will create an ionic current whose magnitude will be proportional to the difference between E_{ion} and the given voltage value. This is something expected, since the bigger the difference, the stronger the electrochemical gradients become, resulting in a stronger influx or efflux of ions. The ionic current created by electrochemical gradients for a given ionic species can be calculated via the Equation (2.5):

$$I_{\text{ion}} = g_{\text{ion}}(V - E_{\text{ion}}), \tag{2.5}$$

where the positive parameter g_{ion} is the conductance of the ion (given in mS/cm^2). Being able to calculate the magnitudes of ionic currents using Equation (2.5) allows us to calculate the net current in a neuron by summing all the ionic currents, information that can be used to understand how electrical signals travel and evolve through neurons. However, the net current also contains an additional element that is the result of the membrane behaving as a capacitor; recall that the membrane separates oppositely charged regions via the lipid bilayer, and since the lipid bilayer is dielectric, it does not allow charges from directly crossing to either side of the cell (unless it is through channels), resulting in an accumulation of charges on either side of the membrane proportional to its voltage. That is precisely what characterises a capacitor. As a consequence, the membrane's behaviour as a capacitor can be described by the Equation (2.6):

$$C = \frac{Q_1}{V} = -\frac{Q_2}{V}, \tag{2.6}$$

where C is the capacitance (given in μF/cm^2); and Q_1 and Q_2 the electric charge stored on either side of the cell, respectively. Therefore, by operating on Equation (2.6), we have that

$$CV = Q \Rightarrow \frac{d}{dt}(CV(t)) = \frac{dQ}{dt}(t) \overset{\text{Def.}}{=} I(t) \xRightarrow{C \text{ is constant}} C\frac{dV}{dt}(t) = I(t).$$

This implies that due to the nature of the membrane voltage as a capacitor, the change in voltage over time (caused by electrochemical gradients) results in a current, known as the **conductive current**. Therefore, the net current on a neuron can be described by the Equation (2.7):

$$I(t) = C\frac{dV}{dt}(t) + \sum_{\text{ion}\in\text{Ions}} I_{\text{ion}}(t). \tag{2.7}$$

Remark. Since we are interested in the dynamics of the membrane, and in order to apply many tools dynamical systems offer, we will be working with an alternative formulation for Equation (2.7), Equation (2.8):

$$C\frac{dV}{dt}(t) = I(t) - \sum_{\text{ion}\in\text{Ions}} I_{\text{ion}}(t). \tag{2.8}$$

This is a fundamental formula that describes the evolution of electric signals in neurons, and it contains many interesting properties. By applying Kirchoff's first law [16], unless there is an external current injected into the system, the net current I becomes zero, meaning that the sum of the ionic current and the capacitive current are equal in magnitude but opposite in sign; that is, both cancel each other out. However, if any external current is injected, we have that $I = I_{\text{inj}}$, and this allows us to control how the membrane voltage will evolve over time. We find this to be a very interesting property, as it illustrates how the input data fed to an artificial neural network, along with its parameters (which can be seen as analogous to ionic currents), ultimately determines its output. However, ANNs do not consider that those parameters evolve over time, but it is an interesting starting point.

To conclude, note that since we can calculate the time evolution of the membrane voltage over time via an ordinary linear differential equation, we can use that formula to solve the ODE for V and obtain the value of the membrane voltage for any time step t. Indeed, solving Equation (2.8) results in Equation (2.9):

$$V(t) = V_{\text{rest}} + \frac{I(t)}{g_{\text{inp}}} + \left(V_0 - V_{\text{rest}} - \frac{I(t)}{g_{\text{inp}}} \right) \exp\left(-\frac{g_{\text{inp}}}{C} t \right), \tag{2.9}$$

where V_{rest} is the membrane's equilibrium potential; $g_{\text{inp}} = \sum_{\text{ion} \in \text{Ions}} g_{\text{ion}}$ the total conductance of the ion channels (for the moment, with the objective of understanding the basic concepts, conductance is assumed to be a constant value); and V_0 the initial membrane voltage (at $t = 0$). This equation further emphasises the fact that if $I = I_{\text{inj}}$, then $V(t)$ will evolve so that it stabilises to a value determined by V_{rest} and I_{inj} (these two components can be identified with the parameters and the input data fed into an ANN, respectively), since $\lim_{t \to \infty} V(t) = V_{\text{rest}} + \frac{I_{\text{inj}}}{g_{\text{inp}}}$.

2.2.3 Voltage Clamp and I-V Relations

So far, we have studied the relations between membrane voltage, electrochemical gradients and ionic currents, but there are still interesting aspects we need to address. One of them concerns the current that needs to be injected in order to hold the membrane voltage at a desired value over time (this does not imply that the injected current will be constant, but rather that by modulating the injected current, voltage will stabilise at a desired value such that the current that is injected at that time corresponds to the current that is being analysed). In reality, membrane voltage is not only time-dependent, but also position-dependent. Since this further difficults the analysis of the neuron's computational properties, a procedure known as **clamping** [17] can be done to ensure membrane voltage becomes only time-dependent. Surely, real neurons are not voltage-clamped, so they exhibit very complex dynamics, but many neurocomputational results come from the membrane being clamped, and since many neural models built on top of these theoretical results tie well with the behaviour shown by real neurons, it is typical to analyse neurons from this perspective.

Voltage clamping allows, among others, to study the relation between the injected current and the desired held voltage value, also known as the **steady-state voltage**, for the membrane. This is a relevant research field, since it may explain complex phenomena such as the

membrane's voltage post-spike evolution trends. Therefore, how can we calculate the current that needs to be injected to keep the membrane at a desired voltage level V_S? Using Equation (2.9), we can solve for I_{inj} when t tends to infinity, provided that it results in a fixed value. This is,

$$\begin{cases} \lim\limits_{t\to\infty} V(t) = V_S. \\ \lim\limits_{t\to\infty} V(t) = V_{\text{rest}} + \dfrac{I_{\text{inj}}}{g_{\text{inp}}}. \end{cases} \implies V_S = V_{\text{rest}} + \frac{I_{\text{inj}}}{g_{\text{inp}}} \Rightarrow I_{\text{inj}} = g_{\text{inp}}(V_S - V_{\text{rest}}). \tag{2.10}$$

Remark. From now on, we shall refer to the asymptotic current needed to obtain a steady-state membrane voltage V_S as $I_\infty(V_S)$.

Remark. A simple result that can be inferred from the relation between $I_\infty(V_S)$ and V_S is that if $V_S = V_{\text{rest}}$, then $I_\infty(V_S) = 0$. This is something expected, since injecting no current will drive the membrane voltage to its resting potential over time. This simple result shows the invaluable quantitative information that I-V relations can offer.

2.2.4 Conductance

So far, we have discussed the role ionic currents and membrane voltage play in describing the flow of electricity within the neuron. Those are the main components the basic Equation (2.5) shows, but there is a third component we have not addressed yet: ion conductance. As expressed in Section 2.2.2, we developed all the theoretical results considering a constant conductance, i.e., a non-time-dependent variable, but this is not actually the case, and it further complicates the theoretical development of many concepts. For instance, Equation (2.8) becomes a nonlinear ODE, whose solution is not given by Equation (2.9). In fact, such a nonlinear equation is generally not solvable by analytical methods. As for the closed formula for $I_\infty(V_S)$, although the expression is correct, we now need to account for the asymptotic conductance value when calculating its corresponding steady-state voltage. Consequently, this subsection aims to explore how time-dependent conductances critically impact neuronal dynamics.

First of all, the ionic currents that are generated by the electrochemical gradients for a population or ensemble of identical channels can reasonably be described by modifying Equation (2.5) in this manner:

$$I = \bar{g}\, p\, (V - E), \tag{2.11}$$

where $\bar{g}$ is the maximal conductance of the population; p is the average proportion of channels in the open state; and E is the reverse potential of the current. If the channels are selective for a single ionic species, then Equation (2.11) becomes $I_{\text{ion}} = \bar{g}_{\text{ion}}\, p\, (V - E_{\text{ion}})$, but since we are now dealing with an ensemble of identical channels rather than the ionic currents for a specific ionic species, it could be the case that the channels are selective for many ionic species, thus requiring the general Equation (2.11). We should think of applying the same philosophy as with the Goldman equation. In any case, since many models we will present

assume that each ensemble of channels is only selective for a single ionic species, we shall provide theoretical developments considering I_{ion}-like formulas.

The main question that arises after presenting Equation (2.11) is how p can be calculated. Recalling Section 2.1.2, channels can be either gated or non-gated, thus having a direct impact on the value p takes. For leak channels, $p = 1$, as they do not have gating mechanisms that allow them to be in an open or closed state. However, the case for gated channels is totally different, since they have gating mechanisms that allow them to be in an open state, in a closed state by no activation or in a closed state by inactivation. In this case, the value of p, the proportion of open channels in the ensemble or the probability that a channel is in the open state, can be given by the Equation (2.12):

$$p = m^a h^b, \tag{2.12}$$

where m represents the probability that a channel of the ensemble is activated through its activation gates (it can also be understood as the proportion of activated channels in the ensemble); a represents the number of activation gates in the channel; h represents the probability that a channel of the ensemble is not inactivated; and b represents the number of inactivation gates in the channel. As Figure 2.2 shows, channels may have both activation and inactivation gates that allow the flow of ionic currents through them, but they may also have a different number of both types of gates. Since a channel is open only when all activation gates are open and no inactivation gate blocks the ion flow, Equation (2.12) reflects the combined effect of all gating mechanisms required for the channel to be in its open state.

Remark. The complex relation between activation and inactivation gates allows multiple states in which an ensemble of gated channels can be. For instance, channels can be partially activated if $0 < m < 1$; completely activated if $m = 1$; not activated or deactivated if $m = 0$; inactivated if $h = 0$; and released from inactivation or deinactivated if $h = 1$. Moreover, some channels do not have inactivation gates, i.e., $b = 0$, and hence $p = m^a$. Such channels do not inactivate and can only be in the closed state if the activation gates are closed. That is why currents flowing through these channels are referred to as **persistent currents**.

Things begin to cloud up now that we have to introduce the dynamics of the activation and inactivation variables m and h, respectively. That is because m and h are time-dependent variables that evolve over time in a nonlinear manner. This has to do with the complex relation between membrane voltage and gating mechanisms. For instance, voltage-gated channels activate once the membrane reaches a voltage threshold value, but those activations result in strengthening ionic currents (since more channels are now open), currents which further alter the membrane voltage, and consequently, the number of voltage-gated channels that will open. At the same time, once gated channels are activated, inactivation gates typically close the channels quickly, further altering the net current flowing through the membrane. This is the reason why in real neurons Equation (2.9) does not correctly model membrane voltage evolution over time. In any case, how can m and h be calculated, or at least, by which equations are they characterised?

As for the activation variable m, its dynamics is described by the first-order differential Equation (2.13):

$$\frac{dm}{dt}(t) = \frac{m_\infty(V) - m(t)}{\tau(V)}, \tag{2.13}$$

where $m_\infty(V)$ represents the **voltage-sensitive steady-state activation function**, i.e., the value of m when holding V constant; and $\tau(V)$ the time constant that adjusts the dynamics. Both functions can be experimentally fitted via voltage clamping and analysing convergence rates under certain circumstances. For example, if we consider a model that contains only one type of channel and persistent currents ($b = 0$), irrespective of whether the modeled channels are selective to one or many ionic species and irrespective of the number of activation gates, then $m_\infty(V)$ can be experimentally obtained. By first holding the membrane potential at a hyperpolarised value V_0, we can ensure $m = 0$, and thus, $I \approx 0$. Then, we shall conduct the voltage clamp experiment such that we stabilise the membrane voltage at a desired steady-state value V_S and measure the current $I_\infty(V_S)$ that is being injected to hold the membrane voltage at that value. Since stabilising the membrane voltage results in the dynamics having no effect over time (this is, $\frac{dV}{dt}(t) = 0$ and $\frac{dm}{dt}(t) = 0$, by the definition of steady-state voltage) and the activation variable m taking the value $m_\infty(V_S)$, we can make use of Equation (2.10) to solve it for $m_\infty(V_S)$:

$$I_\infty(V_S) = \bar{g}\, m_\infty^a(V_S)(V_S - E) \Longrightarrow m_\infty^a(V_S) = \frac{I_\infty(V_S)}{\bar{g}\,(V_S - E)} \Longrightarrow m_\infty(V_S) = \sqrt[a]{\frac{I_\infty(V_S)}{\bar{g}\,(V_S - E)}}.$$

As for the inactivation variable h, its dynamics can be described by the first-order differential Equation (2.14):

$$\frac{dh}{dt}(t) = \frac{h_\infty(V) - h(t)}{\tau(V)}, \tag{2.14}$$

where $h_\infty(V)$ represents the **voltage-sensitive steady-state inactivation function**, i.e., the value of h when holding V constant. As with $m_\infty(V)$, $h_\infty(V)$ can also be experimentally calculated under certain circumstances, usually performing voltage clamping. The procedure is similar to the one provided for the activation variable, so we will not provide details regarding the obtention of $h_\infty(V)$.

In any case, both gating variables play a crucial role in shaping the dynamics of the membrane voltage, as they lead to its nonlinear behaviour. As a consequence of this, the evolution of the membrane voltage over time results in complex phenomena such as variable spiking thresholds [18], dependent on the initial voltage and the injected current. We consider this to be a fundamental property when dealing with neural models, for which we will provide further explanations in Section 2.3.1, and it will be an important consideration in our proposal.

2.2.5 The Hodgkin–Huxley Model. Action Potentials

Previous subsections have sought to study how the membrane potential relates to ionic and capacitive currents, as well as to analyse how the conductances show nonlinear dynamics,

giving rise to complex phenomena within neurons. In this section, we will put everything together and provide one of the most important models in computational neuroscience. The Hodgkin–Huxley model [19] of the squid giant axon describes the evolution of membrane voltage and gating variables of the squid giant axon, and allows understanding, among others, how action potentials are generated. Hodgkin and Huxley determined that the squid axon carries three major currents through their corresponding channels: a voltage-gated persistent K^+ current (this is, $b = 0$) with four activation gates; a transient Na^+ current (this is, $b \neq 0$) with three activation gates and one inactivation gate); and a leak current, mostly composed of chloride ions. Therefore, they concluded that the dynamics of the squid giant axon can be determined by the system of nonlinear differential equations shown in Equation (2.15):

$$\begin{cases} C\dfrac{dV}{dt}(t) &= I(t) - \bar{g}_{\mathrm{K}} n^4(t)(V(t) - E_K) - \bar{g}_{\mathrm{Na}} m^3(t)h(t)(V(t) - E_{\mathrm{Na}}) - \bar{g}_{\mathrm{L}}(V(t) - E_{\mathrm{L}}) \\ \dfrac{dn}{dt}(t) &= \alpha_n(V(t))(1 - n(t)) - \beta_n(V(t))n(t) \\ \dfrac{dm}{dt}(t) &= \alpha_m(V(t))(1 - m(t)) - \beta_m(V(t))m(t) \\ \dfrac{dh}{dt}(t) &= \alpha_h(V(t))(1 - h(t)) - \beta_h(V(t))h(t) \end{cases}, \tag{2.15}$$

where

$$\begin{cases} \alpha_n(V) = 0.01\dfrac{10 - V}{\exp\left(\frac{10-V}{10}\right) - 1}, & \beta_n(V) = 0.125\exp\left(-\frac{V}{80}\right). \\ \alpha_m(V) = 0.1\dfrac{25 - V}{\exp\left(\frac{25-V}{10}\right) - 1}, & \beta_m(V) = 4\exp\left(-\frac{V}{18}\right). \\ \alpha_h(V) = 0.07\exp\left(-\dfrac{V}{20}\right), & \beta_h(V) = \frac{1}{\exp\left(\frac{30-V}{10}\right)+1}. \end{cases}$$

Remark. Although Hodgkin and Huxley presented their equations using the $\alpha(V)$ and $\beta(V)$ functions, we will use the standard forms of the gating variables, that is, Equations (2.13) and (2.14). This is because the following equalities hold:

$$\begin{cases} n_\infty(V) = \dfrac{\alpha_n(V)}{\alpha_n(V) + \beta_n(V)}, & \tau_n(V) = \frac{1}{\alpha_n(V)+\beta_n(V)}. \\ m_\infty(V) = \dfrac{\alpha_m(V)}{\alpha_m(V) + \beta_m(V)}, & \tau_m(V) = \frac{1}{\alpha_m(V)+\beta_m(V)}. \\ h_\infty(V) = \dfrac{\alpha_h(V)}{\alpha_h(V) + \beta_h(V)}, & \tau_h(V) = \frac{1}{\alpha_h(V)+\beta_h(V)}. \end{cases}$$

However, how is it that the Hodgkin-Huxley model explains complex phenomena as action potentials? Well, note that when $V = V_{\mathrm{rest}}$, then $C\frac{dV}{dt}(t) = 0$, which means that all inward and outward currents balance each other so that the net current is zero. This is a

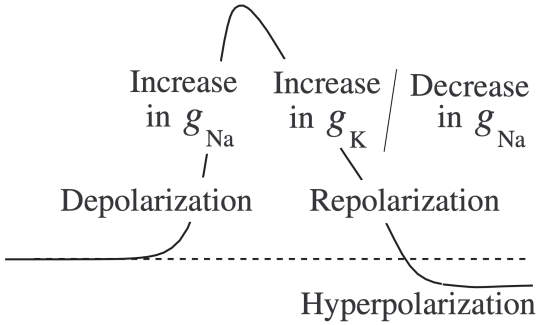

Figure 2.3: Diagram that summarises the action potential mechanism in the Hodgkin-Huxley model. It clearly explains how a strong depolarisation leads to a chain reaction that is then reverted via the efflux of potassium ions and the inactivation of sodium channels. This process results in a hyperpolarisation of the membrane due to the slow progress of variable n before reaching to the equilibrium potential V_{rest}. Image adapted from [6].

resting state, since injecting a small current produces a small depolarisation of the membrane that triggers the voltage back to V_{rest}, without any further effect, as depolarisation is not strong enough to open voltage-gated channels.

In contrast, the case for strong depolarisation of the membrane is quite different, as this triggers a chain reaction that results in what is known as an action potential or spike. This is because strong depolarisation increases activation variables m and n (from 0 to 1, gradually), and decreases the inactivation variable h (from 1 to 0, gradually). On the one hand, both sodium and potassium activation gates start to open, resulting in a stronger influx of sodium and efflux of potassium, respectively. On the other hand, sodium channels begin to inactivate through their inactivation gates. However, the key point to understanding that this process leads to an action potential is that $\tau_m(V)$ is relatively small compared to $\tau_n(V)$ and $\tau_h(V)$, leading to a faster opening of voltage-gated sodium channels and a slower inactivation of these channels. As a result, the membrane undergoes progressively stronger depolarisation, since sodium influx raises the membrane potential before the efflux of potassium and the inactivation of the sodium channels cause voltage values to decrease. However, little by little, the slower gating variables catch up; variable $h \to 0$ causes the inactivation of the Na^+ current, while variable $n \to 1$ causes the slow activation of the outward K^+ current. As a result, they repolarise the membrane potential toward V_{rest}, completing a cycle corresponding to a spike. This process is summarised in Figure 2.3.

Remark. When V approaches V_{rest} during repolarisation, the values of $\tau_n(V)$ and $\tau_h(V)$ are still relatively large, which implies that the outward K^+ current continues to be activated even after the action potential downstroke. This causes V to go below V_{rest} before stabilising around the resting potential, as shown in Figure 2.3, causing a phenomenon known as **afterhyperpolarisation** [20].

2.3 Dynamical Systems in Neuroscience

Much of the content presented in this work focuses on the time evolution of key neuronal variables, such as membrane voltage and activation and inactivation variables. Their dynamics is described by differential equations, and analysing them is crucial to understanding neuronal mechanisms. Since much of the required qualitative analysis can be conducted using tools from dynamical systems theory, this section aims to introduce the fundamental concepts of dynamical systems, explain their relevance to neural models, and highlight how they provide valuable insights for qualitative analysis.

To begin with, dynamical systems are systems of equations that describe the time dependence of their state variables. In our context, since we are working with continuous variables, we shall refer to continuous dynamical systems, which are systems of differential equations whose state variables are time-dependent and their derivatives with respect to time are given by functions of the state variables. This is, they are systems of differential equations of the form shown in Equation (2.16):

$$\begin{cases} \dfrac{dx_1}{dt}(t) = f_1(x_1(t), x_2(t), \ldots, x_n(t)) \\ \dfrac{dx_2}{dt}(t) = f_2(x_1(t), x_2(t), \ldots, x_n(t)) \\ \quad \vdots \\ \dfrac{dx_n}{dt}(t) = f_n(x_1(t), x_2(t), \ldots, x_n(t)) \end{cases} . \tag{2.16}$$

Each function $f_i : A \subseteq \mathbb{R}^n \to \mathbb{R}$, $i = 1, ..., n$, represents the relation between the state variables and the time evolution of the i-th state variable. Therefore, the characteristics of f_i will determine the nature of the i-th differential equation (whether it is linear or nonlinear, for instance).

Dynamical systems contain various properties, components and characteristics that can be analysed to draw insightful conclusions. One such component is the **phase space**, which is the set of points representing all possible combinations of state variables that a dynamical system can occupy at any time, even if the dynamics does not allow some states to be occupied. For example, if a dynamical system is defined with the state variables m, n and h of the Hodgkin-Huxley model, then the phase space will be $[0, 1]^3$, since $0 \leq m, n, h \leq 1$. Surely, many of those triplets will not actually be possible states due to the dynamics not allowing for some combinations. Therefore, when analysing dynamical systems, the actual solutions of the system of differential equations are used to understand the properties of the system. The set of all possible solutions $S = \{(F_1(t), F_2(t), \ldots, F_n(t)) \mid x_i(t) = F_i(t)\}$ describes the possible trajectories that the states follow under the rules of the dynamical system and can therefore be described by vector fields. However, many of the elements in S correspond to qualitatively similar trajectories. Therefore, the analysis of dynamical systems can often be reduced to identifying which trajectories lead to qualitatively distinct behaviours. The set

of those qualitatively different trajectories is known as the **phase portrait**, and it provides invaluable information on the properties of the system.

In computational neuroscience, studying dynamical systems via their phase portraits, for example, tells us, among others, whether there exists some threshold value for the voltage that will trigger a spike in the long term or the value the membrane voltage will evolve to over time. That is the reason why this subsection aims to provide general notions on dynamical systems and their uses in understanding neuronal behaviour. However, we will only focus on one- and two-dimensional systems, since high-dimensional dynamical systems are difficult and computationally costly to tackle. A proposal of a neural model surely needs to be biologically plausible in our scenario, but it also needs to be computationally feasible. One- and two-dimensional systems accurately capture the qualitative behaviour of many neurons while remaining computationally efficient. Consequently, we will focus on understanding their internal mechanisms to provide a solid foundation for the proposal.

2.3.1 One-Dimensional Systems. Resting States, Topological Equivalence and Biological Implications

In this subsection, we will delve into the geometrical methods of analysis of one-dimensional systems. These are systems that only have one time-dependent variable, and although they lack the capacity needed to explain some of the relevant phenomena within neurons, they still show powerful modelling capacity. Moreover, their mathematical simplicity allows for efficient computational modelling, thus resulting in interesting candidates to build biologically plausible neural models for artificial intelligence.

After discussing the core components regarding the electrophysiology of neurons, how could we approach building a one-dimensional system that encompasses many of the modelling capabilities of, let us say, the Hodgkin-Huxley model? That could certainly serve as a strong starting point, as a compelling example can help make the introduction of relevant geometrical concepts more accessible. A neural model proposal could even be based on such a system. Well, note that the Hodgkin-Huxley model can be reduced to a one-dimensional system when all transmembrane conductances have fast kinetics, i.e., $\forall \epsilon > 0,\ \lim_{t \to \epsilon} m(V) = m_\infty(V)$ (and so for the other gating variables). Surely, gating variables do not immediately reach their asymptotic value, but in practice, kinetics are sufficiently fast that trying to build a one-dimensional model from this supposition is actually reasonable. However, this reduction based on the instantaneous kinetics of gating variables only makes sense when a unique gating variable is considered, because otherwise we are losing all the time-dependent interactions between the gating variables that result in phenomena like action spikes. In summary, we either consider a unique gating variable whose kinetics are instantaneous to build a Hodgkin-Huxely-like one-dimensional model, losing the effect other gated channels have on the membrane voltage but providing a reasonable framework, or we consider all the gating variables with instantaneous kinetics, obtaining a model with greater capacity but less coherence. We shall proceed by considering the former.

In such a scenario, what options do we have to build one-dimensional models? First

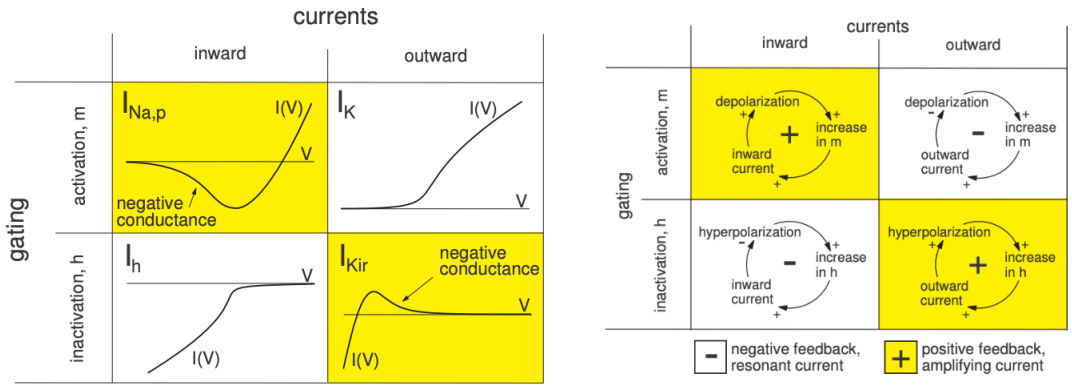

Figure 2.4: Steady-state I-V relations and feedback loops of the one-dimensional models. On the one hand, $I_{\text{Na,p}}$-like models and I_{Kir}-like models show a non-monotonic I-V relation, whereas I_{K}-like models and I_{h}-like models show a monotonic I-V relation. On the other hand, $I_{\text{Na,p}}$-like models and I_{Kir}-like models have a positive feedback loop between the current and the gating variable, whereas I_{K}-like models and I_{h}-like models have a negative feedback loop. Therefore, there is a clear relation between the I-V curve shape and the feedback loop that is generated. Images taken from [6].

of all, since we are considering a single gating variable, we may build either an activation-based model (if any activation variable is considered) or an inactivation-based model (if any inactivation variable is considered). Surely, we can always include leak currents, i.e., $I_{\text{leak}} = -\bar{g}_{\text{L}}(V - E_{\text{L}})$ in our model, since they do not depend on gating variables. Finally, we may build either an inward-current-based model (if $E_{\text{ion}} > E_{\text{L}}$) or an outward-current-based model (if $E_{\text{ion}} < E_{\text{L}}$). Therefore, the four possible combinations result in the following four one-dimensional systems: $I_{\text{Na,p}}$-like models, I_{K}-like models, I_{h}-like models and I_{Kir}-like models. Each type of model features characteristic I–V relationships and feedback loops, as illustrated in Figure 2.4, that make them both understandable and powerful in terms of modelling capabilities.

Therefore, considering everything so far, we could reduce the analysis of one-dimensional systems to the analysis of the four aforementioned models. Since the geometrical tools and analytic procedures are equivalent in the four cases, we shall proceed explaining the most relevant geometrical tools on top of the $I_{\text{Na,p}}$-model, which can be described by the differential equation shown in Equation (2.17):

$$C\frac{dV}{dt}(t) = I(t) - \bar{g}_{\text{L}}(V(t) - E_{\text{L}}) - \bar{g}_{\text{Na}} m_{\infty}(V(t))(V(t) - E_{\text{Na}}). \tag{2.17}$$

One of the most important geometrical properties we can analyse when dealing with dynamical systems is the long-term evolution of its time-dependent variable given a set of initial conditions. This may seem quite uncorrelated with our subject of interest, but it truly gives us invaluable information, among others, on the existence of the well-known threshold value that is related to the firing of spikes. This analysis is one of the easiest and most intuitive we can think of, and that is why it is often introduced in the beginning. However, how is

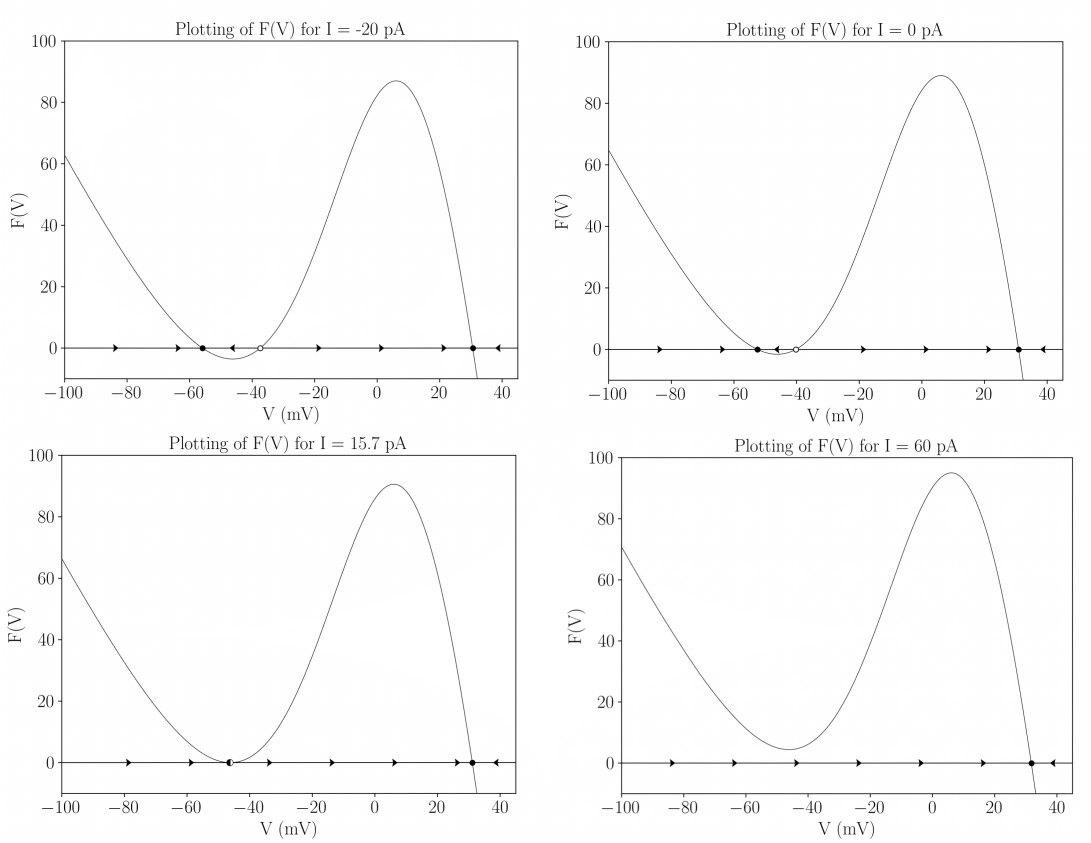

Figure 2.5: Plotting of the time derivative of the $I_{\text{Na,p}}$-model. Shown are the vector fields or phase portraits that describe the evolution of voltage over time, as well as the resting states (stable equilibria are black dots, whereas unstable equilibria are white dots), with the objective to show the importance of these resting states in determining the qualitative behaviour of a system.

it done? The first step consists of plotting the graph corresponding to the time derivative of the state variable, i.e., the plot of the function $F(V(t)) = \frac{dV}{dt}(t)$. Surely, in order to fully define $F(V)$, the values for all the hyperparameters in the equation (like the injected current I in the $I_{\text{Na,p}}$-model) need to be specified, but in many cases it only results in translations on the OY axis, so the process is generally straightforward. For instance, Figure 2.5 shows the plotting of $F(V)$ in the $I_{\text{Na,p}}$-model for a number of injected currents and considering the typical values for conductances, capacitance and other variables [6].

Once the graph of the time derivative has been plotted, the long-term evolution of the state variable can be easily deduced by computing the roots of $F(V)$ and the derivative of F with respect to V. On the one hand, note that if the current state of the time-dependent variable, V_S, satisfies $F(V_S) = 0$, that means that $\frac{dV}{dt}(t \to V(t) = V_S) = 0$, which implies that $\lim_{t\to\infty} V(t) = V_S$. This is, a state value that corresponds to one of the roots of the time derivative of the state variable results in a non-time-dependent value. Such values are known as **equilibria** or **resting points**, since they are time-invariant.

On the other hand, we can also instantly tell how the state variable of the system will evolve over time by looking at the time derivative of F. This analysis, conducted in Appendix A.1, tells us the type of stability each equilibrium point shows, as there is a direct relation between the evolution of the state variable and the equilibrium points, thus reinforcing the idea that equilibrium points provide very useful qualitative information on a neuron's behaviour. In fact, a state V_R is an equilibrium point if it is an eigenvalue of F, i.e., $F'(V_R) = 0$, or if it is a root of F, i.e., $F(V_R) = 0$. Moreover, if $F(V_R - \epsilon) > 0$ and $F(V_R + \epsilon) < 0$ for some $\epsilon > 0$, then V_R will be an attractor, since all the initial states $V(0)$ in the interval $[V_R - \epsilon, V_R + \epsilon]$ will converge to V_R. However, if $F(V_R - \epsilon) < 0$ and $F(V_R + \epsilon) > 0$ for some $\epsilon > 0$, then V_R will be a repeller, since all the initial states $V(0)$ in the range $[V_R - \epsilon, V_R) \cup (V_R, V_R + \epsilon]$ will actually diverge from V_R. In both cases, V_R is an equilibrium point, but we see that it may act as an attractor or as a repeller depending on the time derivative of the state variable. That is the reason why attractors are known as **stable equilibria** and repellers as **unstable equilibria**. Note that knowing the type of equilibrium that each resting point is completely determines the time evolution of the state variable, thus providing an easy way to describe the qualitative behaviour of the system, as in Figure 2.5. In addition, if $F'(V) \geq 0,\ \forall V \in [-\epsilon, \epsilon]$ and for some value $\epsilon > 0$, then $[-\epsilon, \epsilon]$ will be an **attraction domain**, having some stable equilibrium contained or nearby, whereas if $F'(V) \leq 0,\ \forall V \in [-\epsilon, \epsilon]$, then $[-\epsilon, \epsilon]$ will be a **repulsion domain**, having some unstable equilibrium contained or nearby. Note also that by Bolzano's theorem, two stable equilibrium points must be separated by at least one unstable equilibrium (provided that F is differentiable on its whole domain, something we will assume unless specified).

At this point, the reader may ask what the relation between resting states and the well-known threshold is, a question we raised earlier. Interestingly, unstable equilibria play the role of thresholds in one-dimensional bistable systems, i.e, systems having two attractors. This is because unstable equilibria delimit the attraction domains corresponding to the attractors. Therefore, given an unstable equilibrium state value V_U, every state value smaller than V_U will asymptotically converge to the first attractor, whereas every state value greater than V_U will asymptotically converge to the second attractor. Is not this the basic idea behind the threshold? Unless membrane voltage reaches some value, it will not fire, it will return to its resting state. However, if that threshold value is surpassed, that will result in a spike, because voltage will be pushed to its other attractor. Surely, once neurons fire, they do not stay in a firing resting state, but that is explained by the gating variables that return voltage to its resting level, something we are not considering yet. In summary, **subthreshold** perturbations will result in state variables approaching its closest attractor, whereas **superthreshold** perturbations will result in the state variable approaching its furthest attractor. In any case, this is the first time the concept of threshold is thoroughly discussed, a topic we are going to return to soon.

So, analysing the behaviour of a dynamical system, and therefore, the behaviour exhibited by neurons, can also be reduced to the analysis of its phase portrait, as it contains the trajectories, stable and unstable equilibria, etc. that fully define its qualitative behaviour. One of the main advantages of understanding the qualitative behaviour of a neural model or a dynamical system is that we can actually build its **topological equivalent system**,

which is any other dynamical system whose phase portrait is qualitatively equivalent (for a formal definition see [21]). This means that both have the same number of stable and unstable equilibria, limit-cycle attractors, etc. and that their relative positioning is the same. We are really interested in topological equivalent systems when dealing with neural models because they actually maintain all the computational properties of neurons while becoming mathematically more tractable. In the end, our goal is to propose a biologically plausible neural model, but that does not mean that it needs to contain all the biological parameters that neurons exhibit. We are not interested in proposing a fixed experimentally fitted threshold value, but a model that explains phenomena like the action potential. Therefore, if we are able to accurately describe the computational properties of a neuron with a much simpler expression for $F(V)$, that is something we should definitely consider.

The problem with topological equivalent systems is that it is often challenging to find a neural model that is topologically equivalent to another one on the entire state line $\mathbb{R}$. Therefore, a typical approach consists of modelling small neighbourhoods of $\mathbb{R}$ such that finding an equivalent system becomes feasible. In fact, the Hartman-Grobman theorem [22] states that a nonlinear one-dimensional system $\frac{dV}{dt}(t) = F(V(t))$ sufficiently near a non-hyperbolic equilibrium state V_S is locally topologically equivalent to the linear system $\frac{dV}{dt}(t) = \lambda(V(t) - V_S)$, where $\lambda = F'(V_S) \neq 0$. This is a very important result, as it allows one to build neural models that are actually analytically solvable and that preserve the vast majority of the properties of the original system. However, since we need to carefully check whether the local equivalence preserves computational properties, it is, perhaps, not the best alternative when building general-purpose neural models. In any case, this shows that there exist tools that allow one to construct models that mathematically seem simple but that preserve the computational properties of neurons. We shall further comment on the relevance of this idea in Chapter 3.

2.3.1.1 Bifurcations

When presenting Figure 2.5, the reader may have realised that the unstable equilibrium that corresponds to the threshold actually changes depending on the value of the injected current. Not only that, for a sufficiently large input current, the threshold seems to disappear, thus resulting in that any initial state $V(0)$ will end in the neuron firing. This is no coincidence, and corresponds to the final and most advanced qualitative analysis of dynamical systems. Bifurcations refer to the qualitative changes in the phase portrait of a system, and they depend on the initial conditions and parameters of the system. So far, we have seen what resting points are and their relevance within neuronal dynamics, but we must go one step further and analyse how they evolve when initial conditions or the parameters of the system are changed. Before giving any further detail, it is important to highlight that a qualitative change of a system's dynamics does not necessarily imply that the system's evolution will be different, because that also depends on the initial conditions of the system. Therefore, we see that bifurcations are tightly related to the initial conditions and other parameters of the system, which, in our case, are the membrane voltage and the injected current.

Variations in external parameters (such as the injected current) often result in translations

or scalings of $F(V)$ along the OY or OX axis. At first glance, this may not seem significant, as the overall shape of $F(V)$ is generally preserved. However, even a slight modification of these parameters could result in the model losing unstable equilibria points or obtaining stable and unstable equilibria points, since they depend on the roots of $F(V)$. For instance, Figure 2.5 clearly shows that increasing the injected current elevates the graph $F(V)$; the higher the graph of $F(V)$ is, the closer its intersections with the OX axis are, and there is even a point at which both a stable and an unstable equilibrium disappear. Both equilibria coalesce and annihilate each other; a bifurcation has occurred (it is widely known that $I = 15.7$ pA corresponds to the bifurcation value in the $I_{\text{Na,p}}$-model). That is the reason why analysing dynamical systems for a variety of initial conditions and parameters is very important.

However, we can not talk about any type of analysis of bifurcations if we have not yet precisely defined what they are. Since bifurcations play a key role in neuronal dynamics, this subsection aims to provide the general notions behind bifurcations in one-dimensional systems. In general, given a dynamical system that depends on a vector of parameters p (in our case it would only be the injected current), a specific vector of parameters, p_0, in the parameter space $\mathcal{P}$ is said to be **regular** or **non-bifurcation** if $\exists \delta > 0 : \ \forall p_1 \in \{p \in \mathcal{P} \mid ||p_0 - p|| < \delta\}$, the system's phase portrait at $p = p_0$ is topologically equivalent to the phase portrait at $p = p_1$. For example, in the $I_{\text{Na,p}}$-model, it is easy to see that $I = -20$, $I = 0$ and $I = 60$ are all regular values, while $I = 15.7$ is a bifurcation value. Therefore, a system is said to undergo a bifurcation if its current vector of parameters p_1 is a bifurcation value.

Two natural questions that arise after defining bifurcations are, first of all, which types of bifurcations do exist, and second, what their relevant implications in neural modelling are. First of all, only one type of bifurcation can occur in one-dimensional systems: the **saddle-node bifurcation**, which occurs when a stable and an unstable equilibrium annihilate each other. Mathematically speaking, a parameterised one-dimensional system $\frac{dV}{dt}(t) = F(V(t), I(t))$ (we shall only consider injectable current as the only parameter for one-dimensional systems in our context) having an equilibrium point V_S for some value of the parameter $I = I_{\text{inj}}$ is said to be at a saddle-node bifurcation if the following conditions are met [23]:

1. **Non-hyperbolicity.** The eigenvalue of F, when differentiating it with respect to the state variable, at V_S is 0. This is,
$$\lambda = \frac{\partial F}{\partial V}(V_S, I_{\text{inj}}) = 0.$$
2. **Non-degeneracy.** The second order derivative of F with respect to the state variable is non-zero at V_S. This is,
$$\frac{\partial^2 F}{\partial V^2}(V_S, I_{\text{inj}}) \neq 0.$$
3. **Transversality.** F is non-degenerate with respect to the bifurcation parameter I. This is,
$$\frac{\partial F}{\partial I}(V_S, I_{\text{inj}}) \neq 0.$$

Now, which are the biological implications of bifurcations in our context? The most important one is that the concept of threshold is not a fixed thing anymore. The threshold that decides when the neuron shall produce a spike and when it shall continue resting depends on the injected current, which, in our case, we can identify as the input data. Moreover, the common notion of a fixed value for the threshold is no longer a reality, since the unstable equilibrium behind the threshold is a function of the parameters of the system. Surely, having the mathematical definition of the system allows one to calculate whether such a threshold value will exist, and if so, how it will evolve depending on the given values for the parameters.

We find this very important and a major consideration when justifying the characteristics of a neural model. That is why we have conducted an experiment with the objective of showing the dependencies between resting states and parameters of the system that models the neuron. On the one hand, we have simulated the time evolution of the membrane voltage in the $I_{\text{Na,p}}$-model for a set of injected current values and initial membrane voltage values. Figure 2.6 shows the results we have observed for every combination we have worked with, supporting all the theoretical results we have provided throughout this subsection.

For typical values for the injected current ($I \approx 0$), bistability is observed in the system, and therefore, the firing of the neuron will depend only on the initial state of the membrane voltage. The figure in the upper left clearly shows that every initial membrane voltage value below -40 mV will result in the neuron converging into its resting state when no current is injected, whereas every initial voltage value above -40 mV will result in the neuron firing. There is a clear firing threshold value around -40 mV. However, when the injected current reaches a value of 15.7 pA, a saddle-node bifurcation occurs; there still exist both an excited and a resting state, but we can see that the resting state and the threshold are about to be annihilated. Injected currents above 15.7 pA will result in the neuron firing no matter which the initial membrane voltage value is, as the system has now become monostable. Likewise, a very negative value for the injected current will cause the excited state and the threshold to annihilate, after which every initial value for the membrane voltage will converge into the resting state. Note also that the threshold value gradually decreases as the injected current increases. In any case, this experiment clearly shows the importance of the parameterisation of the system in aspects like threshold values.

However, we have taken an additional step. We have conducted another experiment to actually show how the threshold evolves as the injected current increases or decreases. Figure 2.7 shows the numerical approximation of the steady-state voltage for 40,401 combinations (a grid of 201×201, so as to consider 200 discrete values as well as 0) of injected current and initial membrane voltage values in the $I_{\text{Na,p}}$-model. As can be seen, the pinkish wall plays the role of the threshold, because those configurations on one side of that wall result in the neuron resting, whereas every other configuration results in the neuron firing. By paying close attention to the evolution of the pinkish surface, it can be seen that its evolution with respect to the input current is not constant. This is, as the injected current's value increases, the value of the initial voltage that leads to the neuron firing gradually decreases. The figure also shows how the resting membrane voltage of the neuron is also affected by the injected current, since the steady-state voltage value for the resting state gradually increases as current does so.

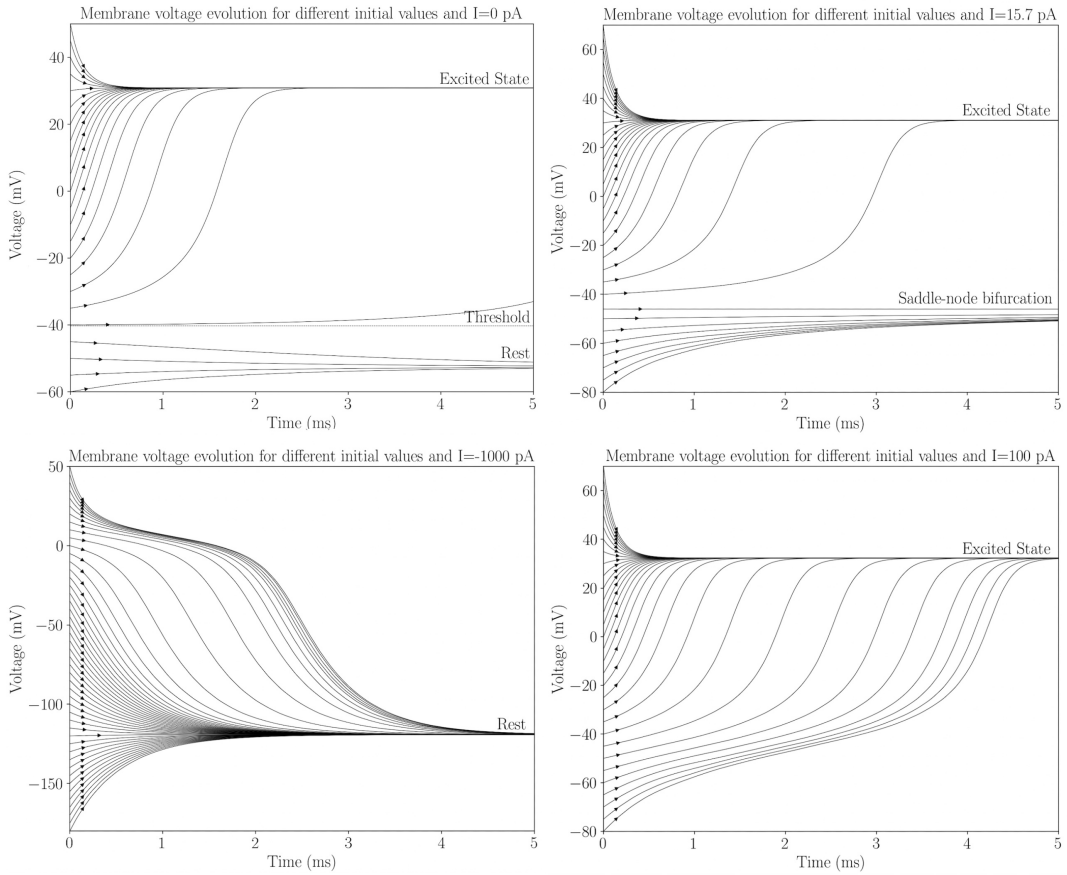

Figure 2.6: Simulations for the time evolution of the membrane voltage in the $I_{\text{Na,p}}$-model. Given a set of initial conditions and parameters, the dynamical system corresponding to the $I_{\text{Na,p}}$-model has been numerically solved via the Runge–Kutta–Fehlberg method [24], and solutions have been plotted as shown. Looking at the plots, it can be easily seen that injected current has a dominant role on the existence of the threshold.

Finally, the saddle-node bifurcation is clearly pictured via the perpendicular pinkish wall that appears near 15 pA, as expected. Therefore, this experiment illustrates the dynamical nature of the threshold, and therefore, something that needs to be considered when implementing a biologically plausible neural model. Having a fixed-threshold model is computationally convenient, as it can be easily implemented, but it does not accurately describe how neurons respond to stimuli.

Remark. Note that all these results correspond to one-dimensional systems, whereas real neurons are much more complex. Therefore, many computational properties have not been covered in this section due to the limitations of one-dimensional systems. In any case, if such "simple" models already reveal complex properties like variable firing threshold values, it is fascinating to imagine everything that could be happening within neurons.

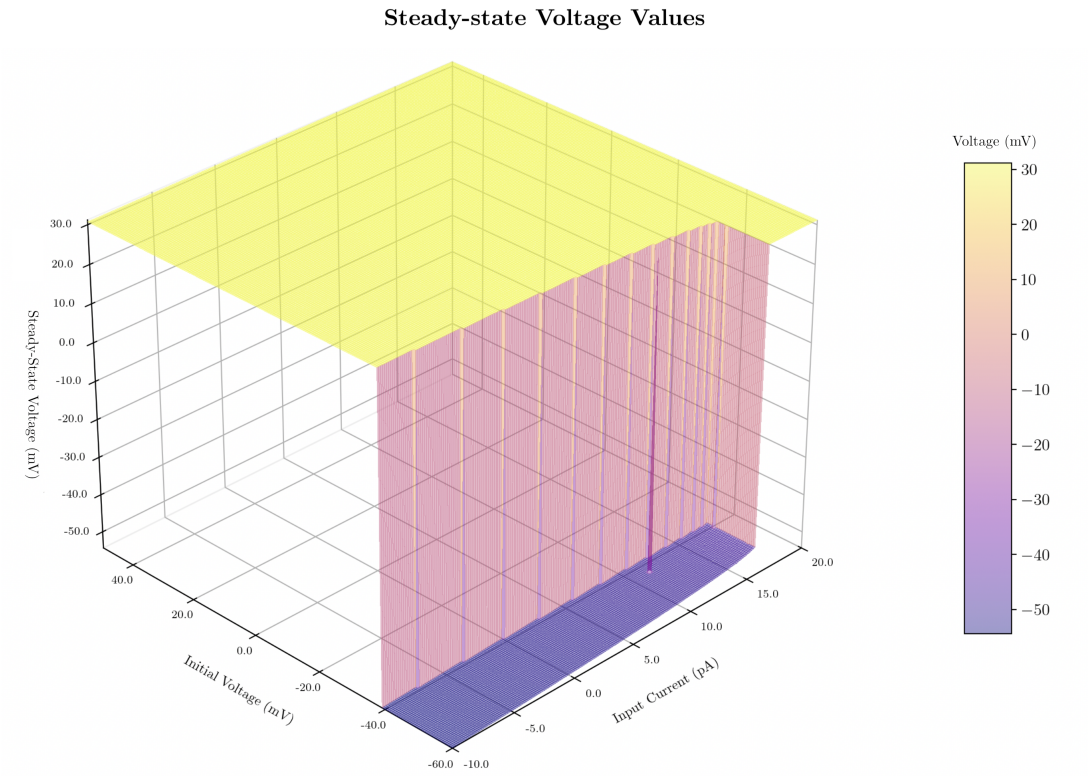

Figure 2.7: Numerical solutions to the $I_{\text{Na,p}}$-model via the Runge–Kutta–Fehlberg method. The figure shows the numerical approximation of the steady-state voltage $V(t \to \infty)$ for 40,401 combinations of initial conditions (injected current I and initial membrane voltage V_0). The figure clearly illustrates the saddle-node bifurcation exhibited by the one-dimensional $I_{\text{Na,p}}$-model as a function of the injected current, as well as the system's bistability for $I < 16$, which can be observed by analysing the steady-state voltage reached from different initial membrane voltages.

2.3.2 Two-Dimensional Systems. Nullclines, Equilibria and Bifurcations

Two-dimensional systems are a natural continuation of the analysis that can be performed to understand how neurons behave. Unlike one-dimensional systems, these provide invaluable information on many phenomena observed in real neurons, aside from reaffirming many properties we have seen so far, such as the spiking threshold being a topological manifold rather than a single immutable value. Therefore, it may be a good idea to analyse their mathematical properties and that is what this subsection will cover. As we did with one-dimensional systems, we will work on top of a specific two-dimensional model to illustrate all the theoretical results that will be provided throughout the subsection, but the analysis can easily be further expanded to other systems. Specifically, we propose to work with the $I_{\text{Na,p}} + I_{\text{K}}$ model [25], a natural extension of the $I_{\text{Na,p}}$ model and whose formulation is given by Equation (2.18):

$$\begin{cases} C\dfrac{dV}{dt}(t) & = I - \bar{g}_{\text{L}}(V(t) - E_{\text{L}}) - \bar{g}_{\text{Na}} m_{\infty}(V(t))(V(t) - E_{\text{Na}}) - \bar{g}_{\text{K}} n(t)(V(t) - E_{\text{K}}). \\ \dfrac{dn}{dt}(t) & = \dfrac{n_{\infty}(V(t)) - n(t)}{\tau(V(t))}. \end{cases} \tag{2.18}$$

We shall see how the $I_{\text{Na,p}} + I_{\text{K}}$ model intuitively explains new types of equilibria, orbits in the phase portrait that lead to the typical periodic spiking activity of neurons and new types of bifurcations. Many results we will comment on can be extended to n-dimensional systems. Consequently, we believe that two-dimensional systems are sufficient to capture the vast majority of the neurocomputational properties of neurons. Therefore, we do not explore n-dimensional systems further, as their increased computational complexity and reduced interpretability reinforce this decision.

Analysis of two-dimensional systems is typically carried out over their planar vector fields, since they provide illustrative depictions of how variables will evolve over time. Given a two-dimensional system of the form

$$\begin{cases} \dfrac{dx}{dt}(t) = f(x(t), y(t)), \\ \dfrac{dy}{dt}(t) = g(x(t), y(t)), \end{cases}$$

where $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ describe the evolution of the state variable $(x(t), y(t))$, the planar vector field of the system is defined as the function $\mathcal{V} : \mathbb{R}^2 \to \mathbb{R}^2$ that assigns to each point (x_0, y_0) a vector whose components correspond to the time derivatives of x and y evaluated at (x_0, y_0), respectively. This is,

$$\begin{array}{rcl} \mathcal{V} : \mathbb{R}^2 & \to & \mathbb{R}^2 \\ (x_0, y_0) & \mapsto & (f(x_0, y_0), g(x_0, y_0)) \end{array}.$$

Note that the vector field gives the gradient with respect to time at every point in the phase plane, and we can use that to intuitively understand how a given state will evolve over time by following the direction of the gradient while constructing the trajectory. Moreover, we can also plot some of the meaningful vectors on the phase plane to perform geometrical analysis that may showcase equilibria, limit cycles, etc. As an example, Figure 2.8 shows the planar vector field corresponding to the $I_{\text{Na,p}} + I_{\text{K}}$ model; parameterisation of the equations has been obtained from [6], so the figure shows a specific planar vector.

There are many insights that Figure 2.8 shows, and we shall address them here. First of all, the phase space is partitioned into four distinct regions that contain qualitatively different planar vectors (without considering the points that lie on the curves that partition the phase plane). Each region contains vectors whose component signs (at least one of them) differ from those in vectors found in other regions. All vectors within a region satisfy the same property, thus providing a clear insight into how any given (V_0, n_0) state will evolve over time. For instance, if (V_0, n_0) is selected somewhere in the upper right region delimited by

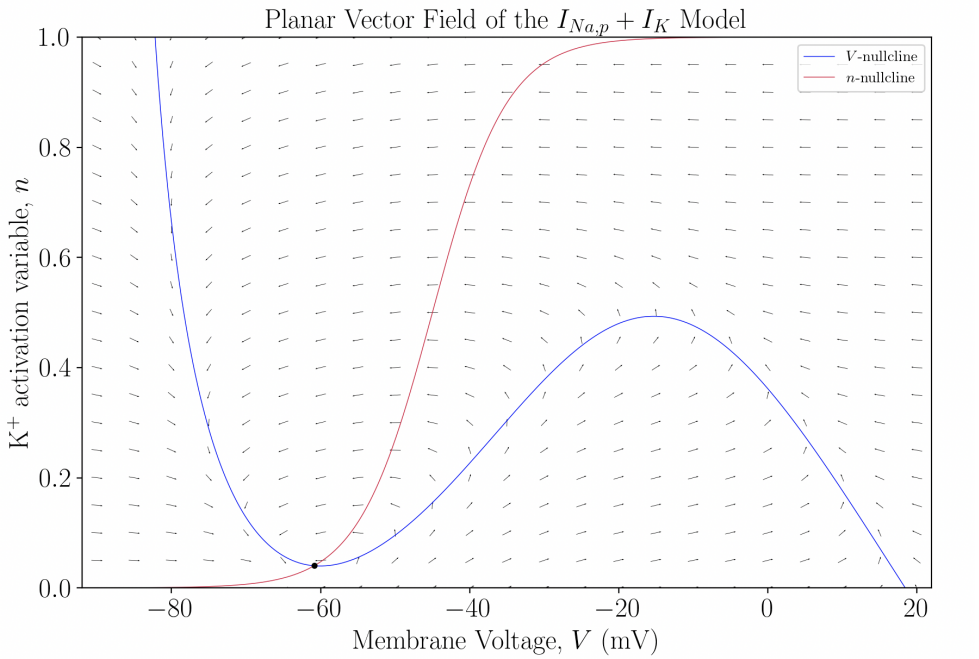

Figure 2.8: Depiction of the planar vector field of the $I_{\text{Na,p}} + I_{\text{K}}$ model. The figure shows the plotting of some of the values calculated via $\mathcal{V}$, together with the V- and n-nullclines and the resting point that appears at the intersection of the nullclines.

the blue and red curves, no matter what the exact value is, voltage will decrease over time, while the activation variable n will increase; the magnitude of the change surely depends on the exact given point, but the qualitative behaviour will be the same. Upon the state reaching a new region, the qualitative evolution of V and n will change, but the possible options are determined by the number of regions into which the phase space can be partitioned, since two different regions cannot provide the same qualitative behaviour for a given state to evolve over time. How is it that the phase plane can be partitioned that way and what is the relevance of this?

It is now that we shall introduce **nullclines**, which are the two curves, $N_V : \mathbb{R} \to \mathbb{R}$ and $N_n : \mathbb{R} \to \mathbb{R}$, whose points have assigned vectors with, at least, one null value within their components. In other words, nullclines are the solutions to the equations $\frac{dV}{dt}(t) = 0$ and $\frac{dn}{dt}(t) = 0$, respectively. From Equation (2.18), they can easily be obtained by solving them for n:

$$\begin{cases} N_V(V) = \dfrac{I - \bar{g}_{\text{L}}(V - E_{\text{L}}) - \bar{g}_{\text{Na}} m_\infty(V)(V - E_{\text{Na}})}{\bar{g}_{\text{K}}(V - E_{\text{K}})} & (V\text{-nullcline}). \\ N_n(V) = n_\infty(V) & (n\text{-nullcline}). \end{cases}$$

Each nullcline partitions the phase space into two regions. All planar vectors that lie on one side of the V-nullcline will have the same sign for the first component (otherwise, the

nullcline would need to be contained in the region, as there would be another point satisfying $\frac{dV}{dt}(t) = 0$, leading to a contradiction), whereas those on the other side of the curve will have a different sign for the first component. Likewise, all planar vectors that lie on one side of the n-nullcline will have the same sign for the second component, whereas those on the other side of the curve will have a different sign for the second component. Therefore, the phase plane is effectively partitioned into four distinct characteristic regions. Note also that states that lie within nullclines will evolve on either voltage or on the activation variable, but not on both. However, there is a special state in which neither variable evolves over time. Note that the intersection of the nullclines has the $(0, 0)$ planar vector assigned. Since there is no evolution of either variable over time, we can conclude that resting states or equilibria can be calculated by solving $N_V(V) = N_n(V)$.

This simple geometrical analysis provides a clear insight into a typical phenomenon found in neurons: spiking activity. An initial state in the lower right region will evolve by voltage increasing and the activation variable n decreasing; upon reaching the upper right region, voltage will start to decrease due to the significant activation of potassium channels; eventually, the state will reach the upper left region, where voltage will continue decreasing, but so will the activation variable; this will lead potassium channels to close little by little, and eventually, upon reaching the lower left region, voltage will begin to increase. This behaviour is explained by the orbit or closed loop that is generated by the planar vectors; any trajectory whose initial state is contained on the loop will evolve as explained, leading to the typical spiking behaviour observed in many neurons. However, note that states near the resting state will gradually converge into it. Therefore, we see that resting states also play a fundamental role in neuronal dynamics. We shall study all those phenomena here.

The first natural question that may arise after describing that equilibria are the intersections of the nullclines is which the nature of those resting states is. In contrast to one-dimensional systems, two-dimensional systems could attract trajectories over some directions whilst repelling others in other directions, so the current outlook regarding types of equilibria is considerably more complex. Not only that, trajectories could be attracted via converging loops, as in Figure 2.8, so it is obvious that we need strong mathematical tools to determine the type of equilibrium each resting state is. However, before delving into the mathematical foundations, we need to precisely define what stability is, as our initial notion of it becomes blurred when introducing a new dimension. On the one hand, we shall define stable equilibria considering the following categories:

1. **Stable equilibria.** An equilibrium state $(x_{\text{eq}}, y_{\text{eq}})$ is said to be stable or Lyapunov stable [26] if any solution to the two-dimensional system $(x(t), y(t))$ with (x_0, y_0) sufficiently close to $(x_{\text{eq}}, y_{\text{eq}})$ remains near it for all time. This is, $(x_{\text{eq}}, y_{\text{eq}})$ is stable if and only if

   $$\forall \epsilon > 0,\ \exists \delta > 0 : |(x_0, y_0) - (x_{\text{eq}}, y_{\text{eq}})| < \delta \Longrightarrow |(x(t), y(t)) - (x_{\text{eq}}, y_{\text{eq}})| < \epsilon,\ \forall t \geq 0.$$

2. **Asymptotically stable equilibria.** An equilibrium state $(x_{\text{eq}}, y_{\text{eq}})$ is said to be asymptotically stable if all solutions starting sufficiently close to $(x_{\text{eq}}, y_{\text{eq}})$ approach it as

$t \to \infty$. This is, $(x_{\text{eq}}, y_{\text{eq}})$ is asymptotically stable if and only if

$$\forall \epsilon > 0,\ \exists \delta > 0 : |(x_0, y_0) - (x_{\text{eq}}, y_{\text{eq}})| < \delta \implies \\ |(x(t), y(t)) - (x_{\text{eq}}, y_{\text{eq}})| < \epsilon,\ \forall t \geq 0 \wedge \lim_{t \to \infty} (x(t), y(t)) = (x_{\text{eq}}, y_{\text{eq}}).$$

3. **Exponentially stable equilibria.** An equilibrium state $(x_{\text{eq}}, y_{\text{eq}})$ is said to be exponentially stable if all solutions starting sufficiently close to $(x_{\text{eq}}, y_{\text{eq}})$ approach it exponentially as $t \to \infty$. This is, $(x_{\text{eq}}, y_{\text{eq}})$ is exponentially stable if and only if

$$\forall \epsilon > 0,\ \exists \delta > 0 : |(x_0, y_0) - (x_{\text{eq}}, y_{\text{eq}})| < \delta \implies \\ |(x(t), y(t)) - (x_{\text{eq}}, y_{\text{eq}})| < \epsilon,\ \forall t \geq 0 \\ \wedge\ \exists a > 0 :\ |(x(t), y(t)) - (x_{\text{eq}}, y_{\text{eq}})| < e^{-at},\ \forall t \geq 0.$$

Note that $(3) \implies (2) \implies (1)$, but not all stable equilibria are asymptotically stable, neither all asymptotically stable equilibria are exponentially stable. Such stable equilibria are referred to as **neutrally stable**, and typically arise when limit cycles appear. In contrast, asymptotically stable equilibria typically arise when foci or stable nodes are present. On the other hand, an equilibrium is said to be unstable if it is not stable. For instability, it suffices to have at least a trajectory that diverges from the equilibrium point, but it could be the case that all trajectories diverge from the resting state, even forming looping trajectories. Therefore, unstable equilibria can be categorised into **unstable nodes**, **saddles** and **unstable foci**. In summary, we need mathematical tools to differentiate neutrally stable equilibria, asymptotically stable nodes, stable foci, unstable nodes, saddles and unstable foci from each other.

Fortunately, we can answer most of the questions regarding types of equilibria by applying the Hartman-Grobman theorem, as it implies that the linearised system described in Equation (2.19) is topologically equivalent to the general two-dimensional system on a neighbourhood of a certain equilibrium (x_0, y_0), provided that it is hyperbolic (we shall define later on what hyperbolic equilibria are and what limitations they pose on determining types of equilibria):

$$\begin{cases} \dfrac{dx}{dt}(t) = \dfrac{\partial f}{\partial x}(x_0, y_0) \cdot (x(t) - x_0) + \dfrac{\partial f}{\partial y}(x_0, y_0) \cdot (y(t) - y_0). \\ \dfrac{dy}{dt}(t) = \dfrac{\partial g}{\partial x}(x_0, y_0) \cdot (x(t) - x_0) + \dfrac{\partial g}{\partial y}(x_0, y_0) \cdot (y(t) - y_0). \end{cases} \tag{2.19}$$

That system can be equivalently written using its matrix form, as Equation (2.20) shows:

$$\begin{pmatrix} \frac{dx}{dt}(t) \\ \frac{dy}{dt}(t) \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x}(x_0, y_0) & \frac{\partial f}{\partial y}(x_0, y_0) \\ \frac{\partial g}{\partial x}(x_0, y_0) & \frac{\partial g}{\partial y}(x_0, y_0) \end{pmatrix} \begin{pmatrix} x(t) - x_0 \\ y(t) - y_0 \end{pmatrix}. \tag{2.20}$$

This topologically equivalent system makes it easier to determine the types of equilibria, since linear algebra tools can be used to draw conclusions. It is the case that we only need to work with the Jacobian matrix of the system, i.e., the matrix whose elements are the partial derivatives of f and g evaluated at (x_0, y_0).

How can linearised systems be used to determine types of equilibria? Appendix A.2 contains the complete derivation that leads to the eigenvalues of the Jacobian matrix being the components that determine most types of equilibria, except for neutrally stable equilibria. Given that the Jacobian matrix J is diagonalisable and that λ_1, λ_2 are its eigenvalues, we have that:

- If $\lambda_1, \lambda_2 < 0$, (x_0, y_0) is a stable node.
- If $\lambda_1, \lambda_2 > 0$, (x_0, y_0) is an unstable node.
- If $\lambda_1 \lambda_2 < 0$, then (x_0, y_0) is a saddle.
- If $\lambda_{1,2} = a \pm ib$ and $a < 0$, then (x_0, y_0) is a stable focus.
- If $\lambda_{1,2} = a \pm ib$ and $a > 0$, then (x_0, y_0) is an unstable focus.

It also turns out that even if J is not diagonalisable, using its Jordan form instead leads to the same conclusions. For example, if J has a single real negative eigenvalue with double geometric multiplicity and its associated eigenvectors do not span the whole vector space, (x_0, y_0) will still be a stable node, but trajectories will approach the node only from the directions the eigenvectors point to, attracting trajectories towards those directions. In such cases, we shall say that (x_0, y_0) is degenerate. Note that degeneracy can not happen when the eigenvalues are complex with a non-zero imaginary part ($\mathrm{Im}(\lambda_i) \neq 0$), since λ_1 and λ_2 are complex conjugates, and degeneracy would imply $\mathrm{Im}(\lambda_i) = 0$.

However, what about neutrally stable equilibria? An intuitive approach would be to think that they arise when the eigenvalues of J are purely imaginary, that is, $\mathrm{Re}(\lambda_i) = 0$ and $\mathrm{Im}(\lambda_i) \neq 0$, since $\mathrm{Re}(\lambda_i) < 0$ results in trajectories spiraling in and $\mathrm{Re}(\lambda_i) > 0$ results in trajectories spiraling out. Therefore, $\mathrm{Re}(\lambda_i) = 0$ would result in trajectories describing concentric orbits in the phase plane around (x_0, y_0); the resting point would be Lyapunov stable but not asymptotically stable, that is, neutrally stable. The problem is that when any eigenvalue of J is purely imaginary, (x_0, y_0) is a non-hyperbolic equilibrium point, and the Grobman-Hartman theorem can not be applied to determine which type of equilibrium it is (it could be a neutrally stable equilibrium, but it could also be a focus, for instance). Therefore, local linear analysis can be used to easily determine the type of equilibrium a resting point is, but only if the eigenvalues are not purely imaginary. However, note that if the system is already linear, since we do not need to apply the Grobman-Hartman theorem, it is true that purely imaginary eigenvalues result in neutrally stable equilibria where trajectories describe concentric orbits around the center point.

As a summary, Figure 2.9 contains the types of equilibria we may find depending on the nature of the eigenvalues of the Jacobian matrix of the topologically equivalent linearised system (considering that the original system is not necessarily linear, thus not being able to determine the nature of non-hyperbolic equilibria). Since the eigenvalues can be computed using the trace and the determinant of the matrix, the figure also contains the classification of equilibria based on the trace and determinant of the Jacobian matrix. It must be said that

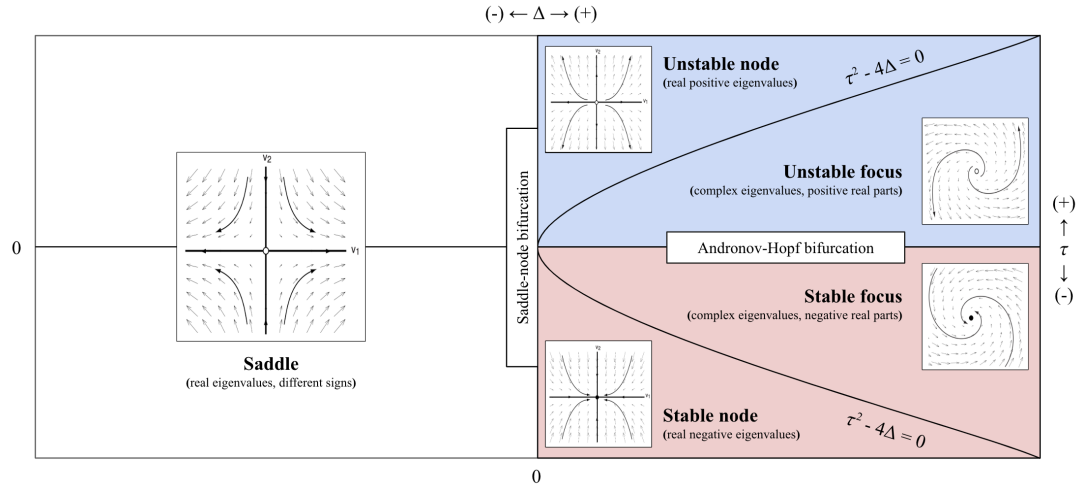

Figure 2.9: Classification of equilibria in two dimensional systems in terms of the characteristics of the eigenvalues of the Jacobian matrix of the topologically equivalent linearised system. Since eigenvalues are related to the trace and determinant of the matrix, both can be used to provide an alternative classification in terms of their properties. Planar vector fields taken from [6].

there exist other types of equilibria in two-dimensional systems that are not contained in Figure 2.9 (as is the case of neutrally stable equilibria), but these are related to non-hyperbolic solutions and they are rare, often appearing in bifurcation scenarios, so this thesis will not cover how they can be determined. [27, 28] contain insightful theoretical notions regarding non-hyperbolic equilibria. In any case, we shall discuss their neurocomputational impact in Section 2.3.2.1.

Understanding types of equilibria is essential in order to justify neurocomputational properties of neurons. This becomes even more interesting in two-dimensional systems with multiple equilibria, such as the FitzHugh–Nagumo model [29, 30], a model that demonstrates that neuronal responses can vary continuously with stimulus strength, including subthreshold and partial-amplitude trajectories, illustrating that **not** all neuron-like dynamics are strictly all-or-none spiking. The FitzHugh-Nagumo model is characterised by the two-dimensional system shown in Equation (2.21):

$$\begin{cases} \dfrac{dV}{dt}(t) = V(t)(a - V(t))(V(t) - 1) - w(t) + I, \\ \dfrac{dw}{dt}(t) = bV(t) - cw(t), \end{cases} \tag{2.21}$$

where w models the recovery variable of an outward current and $b > 0$, $c \geq 0$. In addition, Figure 2.10 shows the planar vector field of the model together with its nullclines for different sets of parameters [6]. The upper planar vector field shows how neurons may not necessarily exhibit all-or-none spiking activity, since initial states whose membrane voltage value is high enough to produce superthreshold oscillations reach different maximum values. In addition, the lower planar vector shows that the threshold is a topological manifold rather

than a single value, as initial solutions that share the same membrane voltage value may result in the neuron spiking or returning to its resting state, clearly determined by the two stable equilibria. In fact, it is possible to calculate the curve that defines the threshold in this case, as it is the curve defined by the pair of trajectories that converge into the saddle equilibrium, denoted as ∘. Those pair of trajectories, known as **separatrices**, partition the phase plane into two attraction domains, each corresponding to one of the stable equilibria. Therefore, if two initial states of the system belong to different attraction domains, even if their membrane voltage value is the same, they will get attracted to different attractors, thus resulting in qualitatively different behaviours. This property needs to be, at least, considered when modelling biologically plausible neural models.

To end with the geometrical analysis of two-dimensional systems, we need to discuss how some types of trajectories that arise lead to explaining phenomena such as periodic spiking behaviour. On the one hand, two-dimensional systems may exhibit trajectories that form closed loops, as is the case with the trajectories around a center equilibrium. Such trajectories are known as **periodic trajectories** and they can be mathematically described as follows:

$$(x(t), y(t)) \text{ periodic } \iff \exists\, T > 0 : \forall t > 0, (x(t), y(t)) = (x(t+T), y(t+T)).$$

The minimal T for which the previous equality holds is called the **period** of the periodic trajectory. Notably, there exist some trajectories that may become periodic from a value t_0 onwards in some systems. These are typically found in bistable systems where an unstable node and a **limit cycle** coexist. However, what are limit cycles? Limit cycles are special periodic trajectories that are found in isolation, rather than in a continuum of periodic trajectories (as in centers), and they play a key role in determining neuronal dynamics. This is because limit cycles can be either attractors (asymptotically stable), if trajectories with initial points sufficiently near the cycles approach them as $t \to \infty$, or repellers (unstable), if trajectories with initial points sufficiently near the cycles diverge from them. This ties well with stable limit cycles containing an unstable node, as for trajectories to converge into the cycle, they must diverge from the inner region delimited by the curve, which means there must be some repeller somewhere inside. As for unstable limit cycles, note that if trajectories diverge from them, there must be some attractor inside the cycle (either some other stable limit cycle or a stable node). This property, supported by the Poincaré-Bendixson theorem [31] and the Poincaré–Hopf theorem [32], shows that two-dimensional systems may explain periodic spiking phenomena in neurons, a property that can not be demonstrated using one-dimensional systems.

As an example, Figure 2.11 shows the phase portrait of the $I_{\text{Na,p}} + I_{\text{K}}$ model together with its nullclines, the limit cycle originated around its unstable focus and the separatrix that defines the threshold. The parameterisation of the model, which differs from the one presented in Figure 2.8, has been taken from [6]. Looking at the characteristic trajectories depicted in Figure 2.11, it is not difficult to infer that the membrane voltage will evolve either into the resting state corresponding to $V \approx -64$ mV or into the limit cycle originated around the unstable focus. Moreover, in the latter case, the neuron will enter a never ending spiking cycle where voltage will evolve following the shape of the limit cycle; voltage will increase to

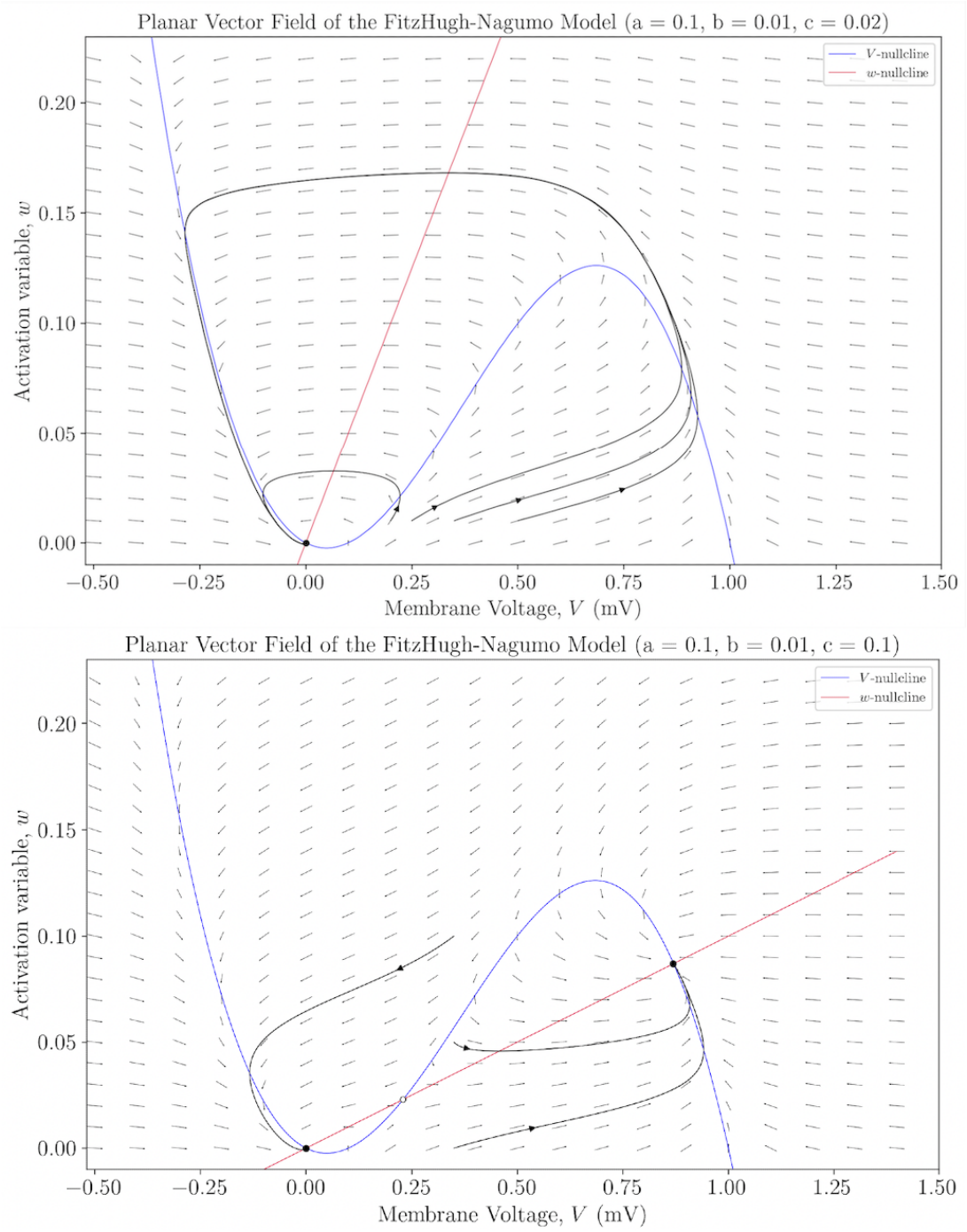

Figure 2.10: Planar vector field of the FitzHugh-Nagumo model for two different set of parameters, along with the nullclines and some characteristic trajectories that show, on the one hand, that neurons may not exhibit all-or-none spiking activity, and on the other hand, that the voltage threshold is not a fixed value.

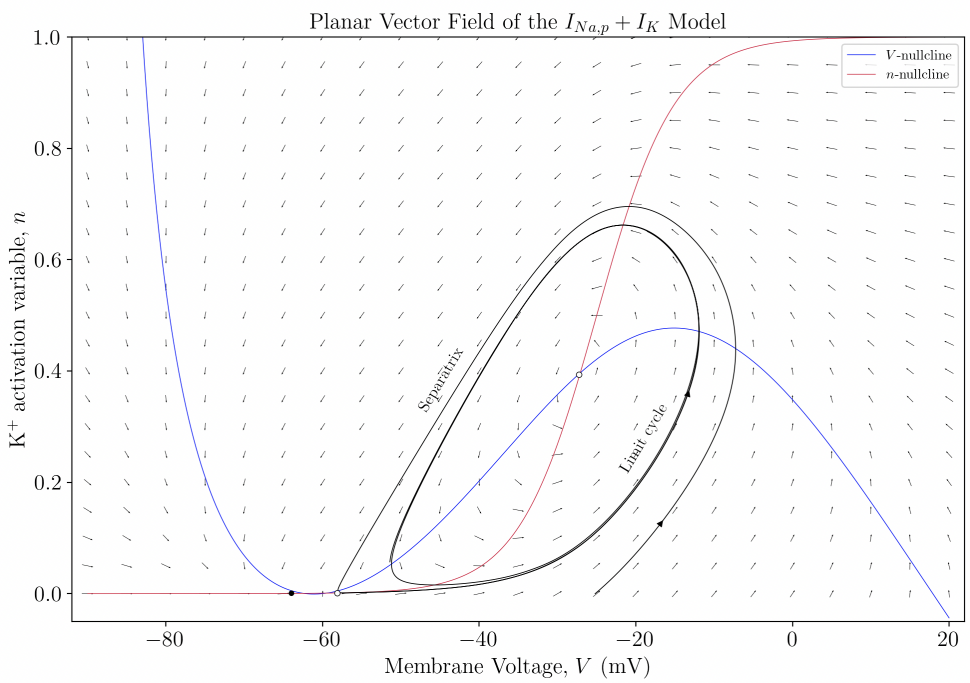

Figure 2.11: Planar vector field of the $I_{\mathrm{Na,p}} + I_{\mathrm{K}}$ model with high-threshold fast K^+ current. Shown are the nullclines, the separatrix and the limit cycle that arises around the unstable focus. Parameterisation taken from [6].

a maximum of approximately -10 mV and decrease to a minimum of approximately -50 mV. Whether an initial state will converge into the resting state or to the limit cycle depends on its location with respect to the separatrix. Every state outside the pseudo-elliptical region delimited by the separatrix curve will converge into the resting state, whereas every state inside that region (except the focus) will converge into the limit cycle.

2.3.2.1 Bifurcations

As we did with one-dimensional systems, it is important to analyse the causes that may lead to qualitative behavioural changes in neuronal dynamics. Bifurcations explain the possibilities for which such phenomena may occur, and this subsection will cover the types of bifurcations we may encounter in two-dimensional systems. Since two-dimensional systems may contain new types of equilibria, the reader might expect a wide range of new bifurcation types to appear, but it turns out that the vast majority are contained within the cases introduced in Figure 2.9. In fact, we know that for hyperbolic equilibria, their nature depends on eigenvalues being real or complex and the sign of their real part. Therefore, most combinations often result in **saddle-node** bifurcations or in **Andronov-Hopf** bifurcations [33].

The idea behind saddle-node bifurcations in two-dimensional systems is analogous to the one developed for one-dimensional systems. Take the $I_{\mathrm{Na,p}} + I_{\mathrm{K}}$ model depicted in Figure 2.11

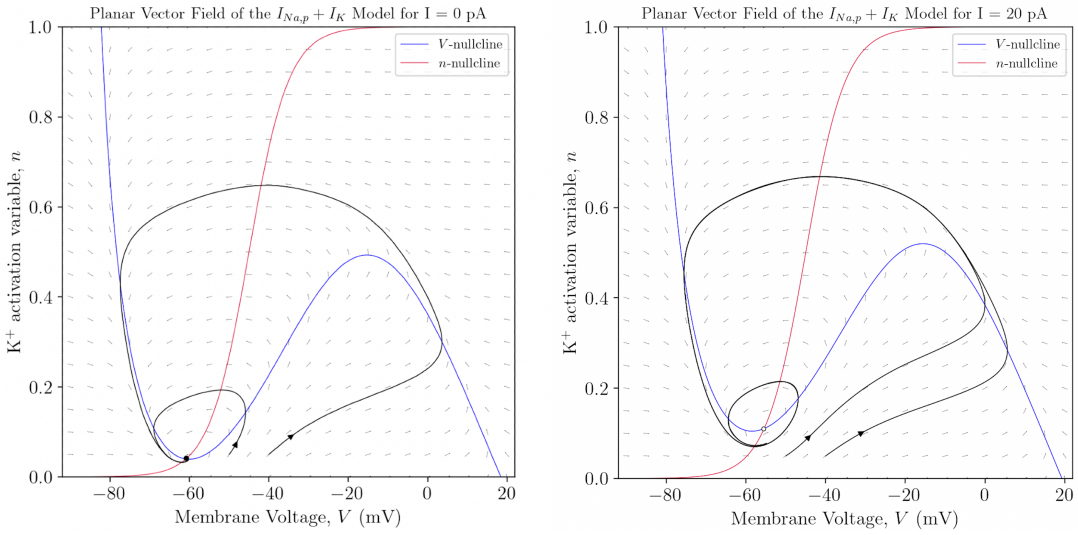

Figure 2.12: Andronov-Hopf bifurcation in the $I_{\text{Na,p}} + I_{\text{K}}$ model with low-threshold K^+ current. Shown are the planar vector fields when $I = 0$ pA and $I = 20$ pA. There is a clear qualitative behavioural change in the phase portrait when the value of I increases or decreases from some value on.

as an example. Increasing the injected current has a quantitative effect on the phase plane of the system, since the V-nullcline is shifted upward. However, the n-nullcline remains unaffected, which results in the stable node and the saddle approaching as I increases. At $I \approx 4.51$ pA, both resting states lie on the same point in the phase space, giving rise to a saddle-node. Mathematically speaking, as both points approach each other, the magnitude of the positive eigenvalue of the saddle starts decreasing, whereas the magnitude of one of the negative eigenvalues of the stable node starts increasing, until, at one point, both become 0, giving rise to a non-hyperbolic equilibrium. It is at this point that both equilibria annihilate each other, which implies that for every $I > 4.51$ pA, every initial solution will converge to the limit cycle. A saddle-node bifurcation has occurred and the notion of the threshold has disappeared.

On the other hand, let us consider the $I_{\text{Na,p}} + I_{\text{K}}$ depicted in Figure 2.8. For small values of I, all states converge to the resting state, which, by the way, is a stable focus. However, when I increases past $I = 12$ pA, the focus loses stability and a limit-cycle attractor appears, as Figure 2.12 shows. The amplitude of the limit cycle continues to grow as I increases. What has happened? Recall that foci have two complex conjugate eigenvalues, which, in our case, have negative real parts (because the focus is stable when $I = 0$ pA). When I increases, so does the real part of the eigenvalues, until it reaches zero at some point. From that point on, the real part will become positive, meaning that the resting point will now be an unstable focus. That is what an Andronov-Hopf bifurcation entails. Andronov-Hopf bifurcations are classified into **supercritical** and **subcritical** depending on whether a limit cycle arises or it disappears, respectively. For instance, Figure 2.12 shows a supercritical Andronov-Hopf bifurcation.

Why may saddle-node and Andronov-Hopf bifurcations be interesting in terms of biological implications? In particular, neurons near a saddle-node bifurcation act as what is known as **integrators**, since they fire all-or-none spikes at arbitrarily low rates (due to the limit cycle fixing the dynamics of spiking states). However, neurons near an Andronov-Hopf bifurcation act as what is known as **resonators**, since depending on the injected current and the initial state, spikes may vary in shape; the **lack of the threshold manifold** in this case causes spiking to begin at a finite minimal frequency, and spike amplitude can vary with input amplitude and timing, giving graded spiking responses. Therefore, perturbations around the resting state evoke damped oscillations at the natural frequency of the system, so the neuron resonates to inputs near that frequency. All of these are insightful properties neurons exhibit that can be explained via bifurcations and analysis of two-dimensional systems.

CHAPTER 3

Modelling Neurons: a Review of Relevant Neural Models

After introducing some of the most relevant concepts in computational neuroscience, together with their biological implications with regards to modelling neuronal behaviour, this chapter will introduce some of the most relevant models that have been used (and even some that are still relevant to the present day) to encompass complex phenomena shown by neurons and provide machines with the ability to learn as we humans do. Indeed, the current growing trend in deep learning raises the question of what the neurocomputational properties of state-of-the-art models are, how they do model neurons and what behind their success there is, motivating the need for this chapter.

However, the reader may have noticed that we have already worked with models in Chapter 2; conductance-based models such as the $I_{\text{Na,p}} + I_{\text{K}}$ model already model several properties of neurons. Therefore, the reader may ask why we have included a separate chapter on neural models when some of them have already been covered. The advantage of the models with which we have worked is that the variables and parameters have well-defined biophysical meanings. We can even measure them experimentally using real neurons. However, things become tedious when trying to measure those biophysical parameters. On the one hand, measurements from different neurons of a certain type need to be considered, as they often vary, even if only slightly. Therefore, averaging the measurements is a typical strategy to parameterise a model, but this results in the model not behaving exactly as experimentally observed. And even if experimental data and theoretical equations come to an agreement, there is no guarantee that the model is accurate from the dynamical systems point of view; it could be the case that the model lacks the capacity needed to characterise all the properties the type of neuron under consideration shows (especially when it comes to bifurcations). Moreover, sometimes we do not need or we cannot afford to have a biophysically detailed model, and instead, simpler models that faithfully reproduce many neurocomputational properties of neurons could be the solution.

That is the reason this section will, on the one hand, review those models that consider many properties introduced in Chapter 2 while being reasonably cheap mathematically and computationally speaking, and, on the other hand, bridge the gap between computational neuroscience and current artificial neural representations employed in machine learning (especially in deep learning) by critically analysing the biological foundations of that field. Deep learning is nowadays everywhere, but it seems that it has become a black box regarding the biological aspects it models, and current development mainly focuses on the optimisation problem deep learning models are linked to. Therefore, it could definitely be an interesting idea to analyse where deep learning stands before complementing it with a biologically inspired model. It must be noted that this section will mainly cover the neurocomputational aspects of models rather than their architectural details. In other words, we assume the reader is familiar with multilayer perceptrons, convolutional and recurrent neural networks and the Transformer architecture, among others.

3.1 Classic Neurocomputational Models

We shall begin by introducing the simplest models employed to mimic neuronal behaviour. These neurocomputational models, typically one- and two-dimensional with a relatively simple formulation, try to encompass some specific properties that we have seen throughout this thesis, providing a clear scheme of the neurocomputational aspects they consider. As such, they have both strengths and weaknesses, which will be discussed in this section.

3.1.1 Leaky Integrate-and-Fire Model

The leaky integrate-and-fire model, first introduced by Louis Lapicque in 1907 [34] and further developed by Richard Stein and Henry Tuckwell [35, 36], considers that neurons feature leak channels and a number of voltage-gated channels that are completely deactivated over time ($m = 0$):

$$C\frac{dV}{dt}(t) = I(t) - \bar{g}_{\mathrm{L}}(V(t) - E_{\mathrm{L}}).$$

Moreover, when the membrane potential reaches a given threshold value, this model mimics the activation of voltage-sensitive currents and the neuron firing an action potential. The process ends with the membrane voltage being reset to a subthreshold value. This structural behaviour is achieved by characterising the leaky integrate-and-fire model by the equations shown in Equation (3.1):

$$\begin{cases} \dfrac{dv}{dt}(t) = b - v(t), & \\ v(t) \leftarrow 0, & \text{if } v(t) = 1, \end{cases} \tag{3.1}$$

where the resting state is $v = b$, the threshold value is $v = 1$ and the reset value of the subthreshold is $v = 0$. The phase portrait of the leaky integrate-and-fire model can be seen in Figure 3.1 for different values of b.

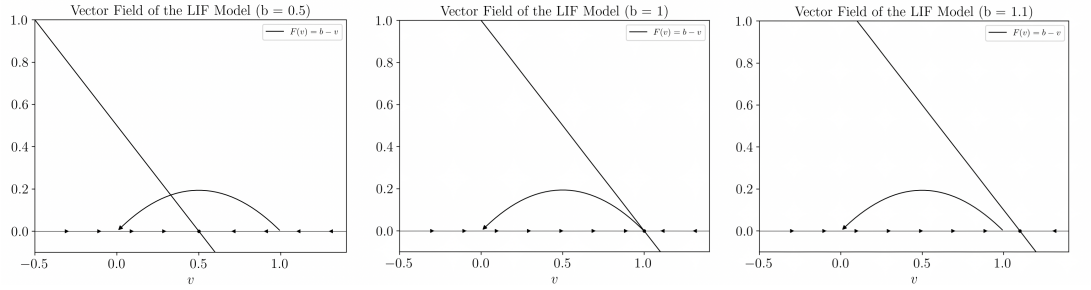

Figure 3.1: Vector field of the leaky integrate-and-fire model for different values of b, together with the function that determines the time evolution of v, i.e., $F(v) = b - v$, and the stable node that arises.

Among the interesting characteristics that the vector fields reveal, we can comment on the behaviour of the system's state when $b < 1$ and $b > 1$. In the former case, the neuron shows excitable behaviour, as small perturbations temporarily raise membrane voltage towards the threshold, but then it decays back to b. In the latter case, though, the neuron shows all-or-none spiking behaviour due to the resting state being above the threshold. Membrane voltage values above the threshold will converge to the stable node, while those below the threshold will continuously reset in a never-ending cycle as they attempt to approach it. This will result in constant-amplitude spikes with a fixed period that can be calculated by solving the ODE (starting from the reset state, that is, $v(0) = 0$) and calculating the time needed for the voltage to reach the threshold value of 1:

$$\begin{cases} v(t) = b(1 - e^{-t}) \\ v(T) = 1 \end{cases} \quad \overset{\text{Solve for } T}{\Longrightarrow} \; b(1 - e^{-T}) = 1 \Longrightarrow T = -\ln\left(1 - \frac{1}{b}\right).$$

Considering such properties, we can conclude that the leaky integrate-and-fire model effectively encompasses several important neurocomputational properties [6]:

1. **All-or-none spikes.** Membrane voltage is reset upon the threshold value is reached. Therefore, all spikes show the same amplitude. This is a characteristic integrator neurons show.

2. **Well-defined threshold.** There exists a fixed value for the threshold ($v_{\text{thresh}} = 1$), there is no ambiguity. This ties well with the characteristics of integrator neurons, where the threshold is determined by the separatrix. Since the leaky integrate-and-fire model is one-dimensional, the separatrix is a 0-dimensional topological manifold (a point).

3. **Relative refractory period.** After the action potential is fired, the neuron becomes less excitable.

4. **Coherent excitation and inhibition.** Excitatory inputs ($b > 0$) bring the membrane voltage closer to the threshold, whereas inhibitory inputs ($b < 0$) move it further from the threshold.

However, this model has some modelling flaws, the most significant being that the transition from resting to repetitive spiking does not occur either via a saddle-node or an Andronov-Hopf bifurcation, but by a mathematical add-on as a piecewise function. Moreover, this model can not exhibit spike latencies because stimuli evoke immediate spikes. In addition, this model is not technically a spiking model, as it lacks any spike generation mechanism; it can not produce regenerative depolarisation of membrane potential caused by the chain reaction of voltage-gated channels opening. Instead, an artificial mechanism is added to simulate the spiking behaviour. In any case, the leaky integrate-and-fire model shows that even simple systems can exhibit robust behaviour.

3.1.2 Quadratic Integrate-and-Fire Model

The quadratic integrate-and-fire model [37] can be seen as an enhancement or refinement of the leaky integrate-and-fire model presented in the previous subsection, especially when it comes to spike generation. The model is characterised by the set of equations presented in Equation (3.2):

$$\begin{cases} \dfrac{dv}{dt}(t) = b + v^2(t), & \\ v(t) \leftarrow v_{\text{reset}}, & \text{if } v(t) = v_{\text{peak}}, \end{cases} \tag{3.2}$$

where v can be understood as the membrane voltage, v_{reset} as the membrane voltage value to which the model will be reset after reaching a certain voltage value and v_{peak} the peak of a spike (rather than a threshold, as was in the previous case). In addition, the phase portrait of the model for different values of b and v_{reset} is shown in Figure 3.2.

Why is it that the quadratic integrate-and-fire model exhibits "better" properties? The key idea is that the differential equation that characterises the model, $\frac{dv}{dt} = b + v^2$, is a topological normal form [38] for the saddle-node bifurcation, which implies that the model behaves as any Hodgkin-Huxley-type model near that bifurcation [6]. Moreover, considering the different combinations of parameters, the quadratic integrate-and-fire model exhibits some properties that closely relate to those addressed in Chapter 2:

- If $b > 0$, there exists no stable state, thus the neuron will enter into a never-ending periodic spiking state, characterised by the differential equation that determines how voltage will evolve, the peak voltage value and the reset voltage value. In such a case, the period of the spiking can be calculated by integrating the ODE:

$$\frac{dv}{dt}(t) = b + v^2(t) \Longrightarrow \frac{1}{b+v^2}dv = dt \Longrightarrow \int_{v_{\text{reset}}}^{v_{\text{peak}}} \frac{1}{b+v^2}dv = \int_0^T dt = T \Longrightarrow$$

$$\frac{1}{\sqrt{b}} \tan^{-1}\left(\frac{v}{\sqrt{b}}\right)\Bigg|_{v_{\text{reset}}}^{v_{\text{peak}}} = T \Longrightarrow T = \frac{1}{\sqrt{b}}\left(\tan^{-1}\left(\frac{v_{\text{peak}}}{\sqrt{b}}\right) - \tan^{-1}\left(\frac{v_{\text{reset}}}{\sqrt{b}}\right)\right).$$

- If $b < 0$, there exists both a stable and an unstable equilibrium. The stable equilibrium corresponds to the resting state, whereas **the unstable equilibrium corresponds to the threshold** (that is why v_{peak} is not referred to as v_{thresh}); all the states to the left of

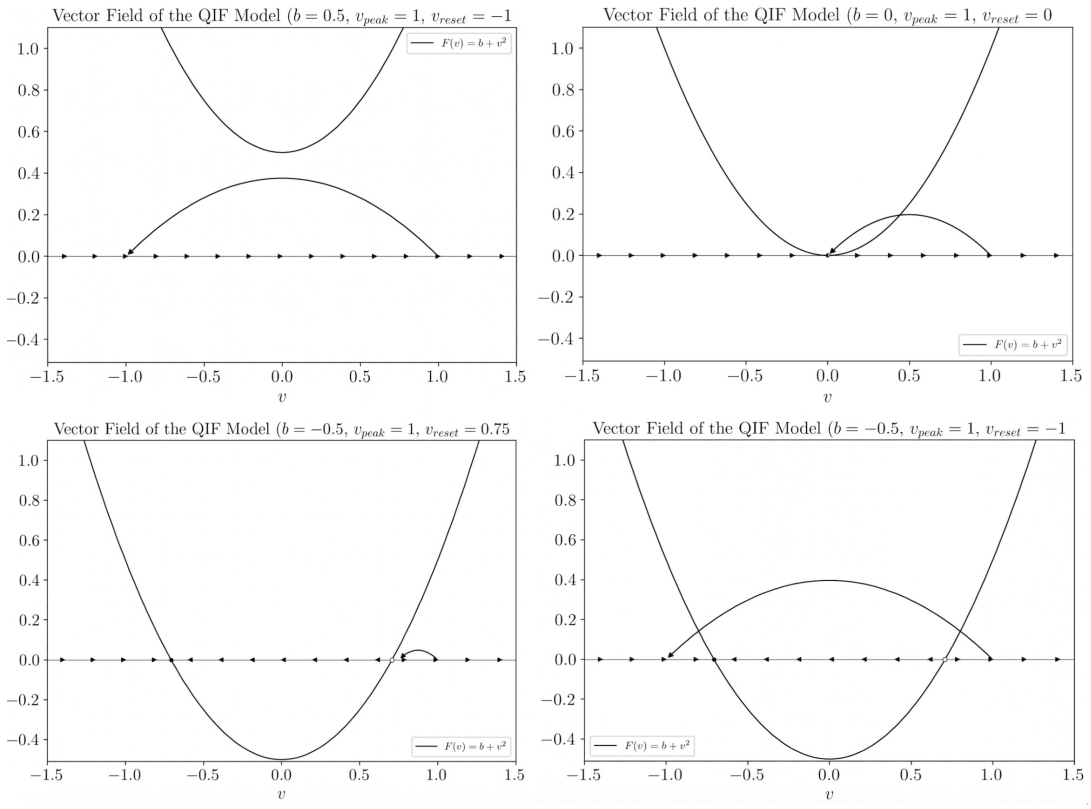

Figure 3.2: Vector field of the quadratic integrate-and-fire model for $v_{\text{peak}} = 1$, different values of b and v_{rest}, together with the function that determines the time evolution of v, i.e., $F(v) = b + v^2$, and the equilibria that arise.

the unstable equilibrium will converge to the resting state, whereas all the states to the right of the unstable equilibrium will initiate an action potential. As for this last case:

- If $v_{\text{reset}} > \sqrt{|b|}$, then the neuron will enter a never-ending spiking state, revealing a coexistence of resting and periodic spiking states. The period of the spikes can be obtained as usual (but with a little adjustment to account for the fact that $b < 0$):

$$\begin{aligned}\frac{dv}{dt}(t) = -|b| + v^2(t) \implies \frac{1}{v^2 - (\sqrt{|b|})^2}dv = dt \implies \int_{v_{\text{reset}}}^{v_{\text{peak}}} \frac{1}{v^2 - (\sqrt{|b|})^2}dv \\ = \int_0^T dt = T \implies \frac{1}{2\sqrt{|b|}} \ln\left(\left|\frac{v - \sqrt{|b|}}{v + \sqrt{|b|}}\right|\right)\Bigg|_{v_{\text{reset}}}^{v_{\text{peak}}} = T \overset{v_{\text{reset}}, v_{\text{peak}} > \sqrt{|b|}}{\implies} \\ T = \frac{1}{2\sqrt{|b|}}\left(\ln\left(\frac{v_{\text{peak}} - \sqrt{|b|}}{v_{\text{peak}} + \sqrt{|b|}}\right) - \ln\left(\frac{v_{\text{reset}} - \sqrt{|b|}}{v_{\text{reset}} + \sqrt{|b|}}\right)\right).\end{aligned}$$

- If $v_{\text{reset}} \leq \sqrt{|b|}$, then voltage will return to its resting state after the action potential has been fired.

All these properties make the quadratic integrate-and-fire model exhibit a more robust performance compared to the leaky integrate-and-fire model. Apart from the properties mentioned for the leaky integrate-and-fire model, this second model is a true integrator model. It exhibits saddle-node bifurcations; it has a dynamic but well-defined threshold (which depends on the value of b, except in the cases where there is no unstable equilibrium, in which case $v_{\text{thresh}} = v_{\text{peak}}$); and spikes are generated with latencies. Therefore, this model is a truly good candidate for modeling integrator neurons. However, it must be noted that the quadratic integrate-and-fire model falls short when trying to model resonator neurons.

3.1.3 Resonate-and-Fire Model

We have seen several options for modelling integrator neurons. The fact that they exhibit robust and intuitive behaviour (with a well-defined threshold value, either a fixed value or not) makes them suitable for deploying large-scale neurocomputational models. As such, it should be no surprise that they have had a considerable influence on neural representations in machine learning. However, Section 2.3.2.1 has shown us neurons can also exhibit resonator-like behaviour, and therefore, we should also study attempts that have been made to model especially these types of neurons.

The resonate-and-fire model [39, 40] can be seen as the two-dimensional extension of the integrate-and-fire model that includes a low-threshold persistent K^+ current. The model can be characterised by the equations presented in Equation (3.3):

$$\begin{cases} C\dfrac{dV}{dt}(t) = I(t) - \bar{g}_{\text{L}}(V(t) - E_{\text{L}}) - W(t), & \\ \dfrac{dW}{dt}(t) \;\; = \dfrac{V(t) - V_{1/2}}{k} - W(t), & \\ V(t) \leftarrow V_{\text{reset}}, & \text{if } V \geq V_{\text{thresh}} \\ W(t) \leftarrow W_{\text{reset}}, & \text{if } V \geq V_{\text{thresh}} \end{cases} \tag{3.3}$$

where W denotes the magnitude of the persistent K^+ current. In addition, Figure 3.3 shows the phase portrait of the model.

There are some interesting things we need to mention regarding the resonate-and-fire model. First of all, an artificial threshold determines when a spike is fired, as reaching the threshold resets the (V, W) state to $(V_{\text{reset}}, W_{\text{reset}})$. Note that the artificial threshold also imposes an all-or-none spiking behaviour for superthreshold states. This may seem contradictory with some of the properties introduced in Section 2.3.2.1, as we commented that resonator-like neurons may not have a clear threshold manifold defined, as well as that their behaviour is not necessarily all-or-none spiking, including subthreshold and damped oscillations of membrane voltage. However, the inclusion of the artificial threshold enables analysing which trajectories lead to a spiking behaviour and which return to the stable focus, which might conceivably be both an effective and efficient strategy when building a computational model that relies on deterministic spiking. Thresholds are widely used in machine learning to determine if an instance should be classified as A or B, or to determine if

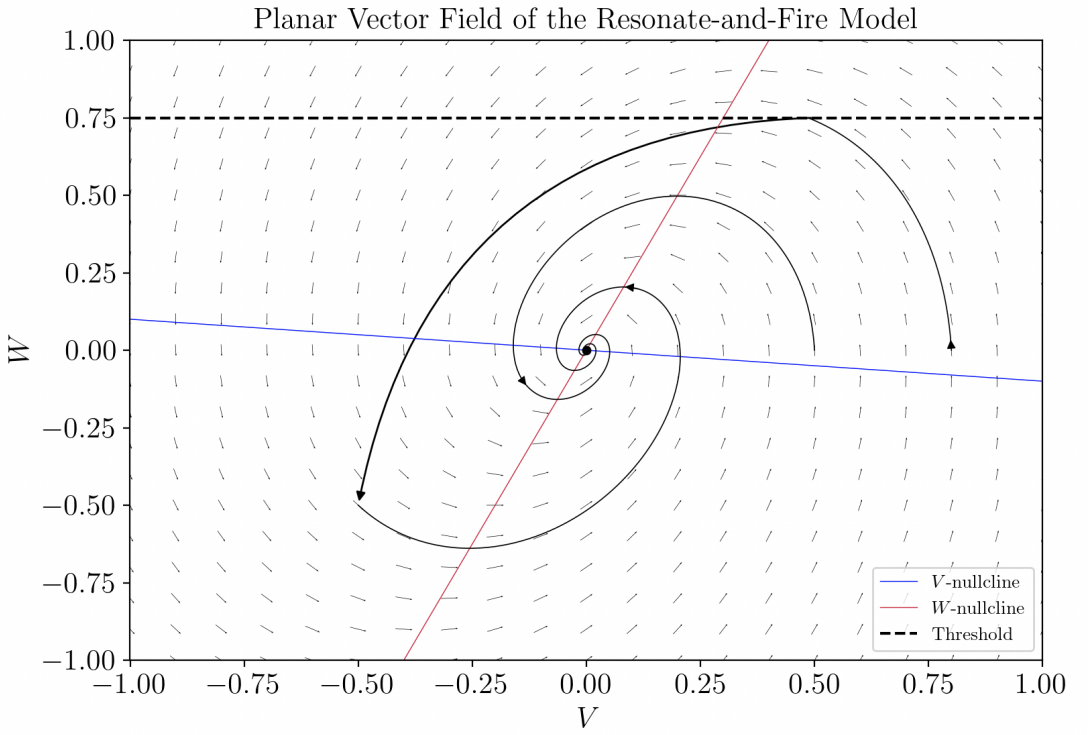

Figure 3.3: Planar vector field of the resonate-and-fire model. Nullclines, stable equilibria and characteristic trajectories are also shown. Parameterisation: $C = 1$, $\bar{g}_{\mathrm{L}} = 0.1$, $E_{\mathrm{L}} = 0$, $V_{1/2} = 0$, $k = 0.4$, $V_{\mathrm{reset}} = -0.5$ and $W_{\mathrm{reset}} = -0.5$.

an instance belongs to cluster A or to cluster B, among others. Therefore, although modelling capacity seems to have been reduced, other interesting features may have been obtained.

This does not imply, though, that the resonate-and-fire model does not include resonator-like properties. In fact, this model illustrates some of the most important features of resonators: damped oscillations, frequency preference (resonance) and postinhibitory spikes. On the one hand, subthreshold states will produce damped oscillations and postinhibitory spikes as those states are attracted towards the focus. Note also that trying to move any state away from the focus via external impulses may result in unexpected behaviour: if a given state has completed a cycle around the focus and is located in the first quadrant of Figure 3.3, raising V or W will effectively move the state away from the focus; but if the state has completed half a cycle around the focus and is located in the fourth quadrant, the same external input will bring the state closer to the focus. This demonstrates that the model prefers inputs with certain frequencies when it comes to its excitation; the model resonates at some frequencies. In summary, the resonate-and-fire model accurately explains several phenomena resonant neurons exhibit while maintaining computational efficiency.

We have already seen integrate-and-fire and resonate-and-fire models. Each of these takes into consideration some types of neurons, but which is better? That is a typical question that

is asked when dealing with models (and not only in computational neuroscience, but also in many disciplines where modelling is fundamental). The answer depends on the application to be developed, but in any case, we should understand that both models complement each other. Note that both the leaky integrate-and-fire and the resonate-and-fire models are linear, and therefore, they can be useful to prove mathematical results and analyse neuronal dynamics. However, they all have their modelling and computational strengths and flaws, so when proposing any computational model for a neuron, we should critically analyse its applicability, the extent to which it is a biologically plausible model, large-scale deployment feasibility, etc. Current research on machine learning mostly focuses on practical results rather than on its foundations, so it might be a good idea to stop for a moment and analyse where we are.

3.1.4 Canonical Models

To end with the first part of the third chapter, we shall discuss an idea introduced in Section 3.1.2: representing a wide array of complex neurocomputational models with a general, often simpler, mathematical formulation. The quadratic integrate-and-fire model is a topological normal form for the saddle-node bifurcation, which makes it suitable to model complex models like Hodgkin-Huxley-like models in an efficient way. Such "special" models offer qualitatively advantageous properties and play a major role in computational neuroscience. As mentioned in the introduction of this chapter, it is generally very difficult to precisely determine the parameters that govern the dynamics of a system, as they can vary slightly from neuron to neuron, even among neurons of the same type. Averaging the measurements for a number of neurons might result in losing modelling capacity, as models may no longer correctly describe the dynamics of any of the neurons considered. Therefore, a shift in perspective is needed to address these issues. We have conveyed the main idea behind this shift in perspective, namely, the pursuit of mathematically simple models that can exhibit behaviour similar to their more complex counterparts, but we need to generalise this idea.

Indeed, it is often more productive to consider families of neural models that share a common property, such as all being integrators, and to select a representative model to describe the neurons of interest. In this manner, if there exists a model in the family whose expression is simpler than the others, it could definitely offer robust modelling capacity while being computationally not that expensive. However, how can that be done? The canonical model approach attempts to address this. A model is said to be **canonical** for a given family of models if there exists a piecewise continuous change of variables that transforms any model from the family into this one. Since the change of variables does not need to be invertible, simple lower-dimensional models are often good candidates for canonical models. The key benefit of canonical models is that they retain many important properties of the family while being more tractable. For instance, if the canonical model contains multiple attractors, then all models in the family will also have multiple attractors [6]. Therefore, canonical models allow us to analyse neurocomputational properties that are universal to their corresponding families.

The main issue related to canonical models is that, as far as we know, there exists no universal procedure to derive such models given a family of neurocomputational models.

Progress has been made in fields like local normal-form reduction [41], data-driven model discovery [42] and neural ordinary differential equations [43], but it remains a tough challenge. For instance, the canonical model of a system near an equilibrium state corresponds to its topological normal form at that equilibrium [23] (as it happens with the quadratic integrate-and-fire model), but in general, it is not easy to work on canonical models. In any case, it is interesting to consider this aspect when addressing biologically plausible neural models, as even the simplest models can contain high and robust modelling capacities. We shall discuss this in future chapters.

3.2 Deep Learning: the New Neural Paradigm

After reviewing classic neurocomputational models, with a strong focus on their tight relation with computational neuroscience, the second part of this chapter will cover the last theoretical aspects that need to be considered before working with biologically plausible neural models in artificial intelligence. This section will also serve as a transition towards our goal. Although we have seen how neurons can be modelled by different sets of equations, we should analyse how these have been and currently are considered when developing neural models in artificial intelligence. Moreover, since neurons play a prominent role in machine learning, and especially in deep learning, the main field within artificial intelligence that relies on artificial neural networks and that has seen considerable success in recent years, it might be a good idea to focus the analysis on this area. As a consequence, this section will describe the neurocomputational aspects behind some of the fundamental models in deep learning, providing the key points to bridge the gap between this field and computational neuroscience. A shift in perspective will be needed, as deep learning is not that much about planar vector fields, nullclines, electrophysiology, etc., but about an abstraction of those concepts that allows machines to behave as we humans do, at least to a great extent. In any case, it will be important to establish certain connections between both fields.

3.2.1 The Perceptron and Multilayer Perceptron

It would be hard to talk about deep learning without mentioning Frank Rosenblatt's perceptron. Its architectural configuration, commented on Chapter 1, has served as the basis for building artificial neurons, neural networks and deep learning on top of it. Convolutional neural networks, recurrent neural networks, transformers, even diffusion models, all integrate some components of the perceptron (if not the entire perceptron), hence the importance of considering it. The core equation behind the perceptron is a linear combination $w_1x_1 + \cdots + x_nx_n + b$, where $w_1, \ldots w_n$ are referred to as the weights of the neuron and $x_1, \ldots, x_n$ as the input features, to which the Heaviside step function H is applied to account for the complex interactions between the weights and the input data, as shown in Equation (3.4):

$$\hat{y} = H\left(\sum_{i=1}^{n} w_i x_i + b\right) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i x_i + b > 0 \\ 0, & \text{else} \end{cases}. \tag{3.4}$$

This is then integrated into a framework where the model is optimised to minimise prediction or regression error of a dataset via gradient descent and effectively make it possible for machines to learn. There are thousands of details that could be mentioned to account for all the situations in which the perceptron could be used, the types of learning in which it can be applied, etc., but here we are interested in understanding the biological implications of its core equation. Which are the biological assumptions that lead to the conceptualisation of this model?

The greatest source of inspiration for answering this question may be encountered in Frank Rosenblatt's article of 1958 where he proposed and introduced the perceptron [1]. Rosenblatt himself describes the organisation and conceptualisation of the perceptron on top of the following three rules (generalised and summarised not only for photo-perceptrons as in the article):

1. There exists a set S that contains a collection of sensory units that can be impinged by stimuli and that respond in an all-or-none manner, as integrator neurons do. Rosenblatt addresses the spatial location of the sensory units, something that is not relevant yet but that is tightly related to how convolutional neural networks are conceived.

2. Stimuli are transmitted to a set of association cells, referred to as A-units. Each A-unit receives impulses from a set S_A of sensory units, known as **origin points**. The key property of origin points is that they may transmit either **excitatory** or **inhibitory** impulses, something that ties well with what we have seen in Chapter 2 when discussing how different types of currents may lead the neuron to excitatory or resting states. The next key idea is that all these inputs are linearly combined to obtain the algebraic sum of all the impulses, which could be understood as the total injected current, also discussed in Chapter 2. In other words, impulses are **integrated**. Now, if the algebraic sum reaches a threshold value θ of the A-unit, then the unit will fire in an all-or-none basis. This ties quite reliably with the integrate-and-fire model we have covered in Sections 3.1.1 and 3.1.2, with a fixed value for the threshold, integration of impulses and all-or-none spiking. However, so far, all the parameters will be fixed.

3. Firings of A-units are transmitted into a set of response units, referred to as R-units. Each R-unit has a set of associated A-units, known as the **source-set**. R-units behave much like A-units, with the exception that R-units are connected in both directions with A-units. This allows for having both excitatory and inhibitory feedback connections between both types of units, a property that is used to establish the learning mechanism of the model. Whenever a R-unit gives an all-or-none response, it will send both an excitatory response to its source-set and an inhibitory response to the complement of its source-set, therefore inhibiting other R-units and resulting in a mutually exclusive system where the neuron that fired first determines the outcome. This organisation is suitable for learning, since if such a system is capable of learning, it should be able to modify the A-units or their connections in such a way that stimuli of one type trigger a stronger impulse in a R-unit (and consequently, on its source-set) than in the other ones.

The conceptualisation of the perceptron, although complex, has nowadays been reduced to R-units, as it is generally agreed that the perceptron is the unit of computation that calculates a linear combination and produces a spiking response via a nonlinear function. Therefore, A-units merge with sensory units in what is known as the input layer. Moreover, inhibitory responses between R-units also disappear, but these simplifications still preserve some core principles of integrate-and-fire-like neurons. A mathematically robust learning algorithm, based on gradient descent and statistics, suffices for achieving those inhibitory responses in some sense, thus the idea of the original perceptron not being that distorted.

In addition, a design as the one provided by Frank Rosenblatt opens the door for a reasonably straightforward implementation of neural networks, as stacking R-units in layers and integrating the responses from previous layers makes it possible to adapt the perceptron into a multilayer perceptron without the need to modify the architecture or the learning scheme. Certainly, the design and computations become more complex, but the underlying mechanisms remain the same.

We have discussed many biologically relevant properties of the perceptron, mainly its relationship with integrate-and-fire-like neurons, but this modelling of the neuron lacks several important properties that determine the dynamics of neurons, apart from those introduced in Chapter 1. On the one hand, there is **no time-dependence**, meaning that each input will produce the same output no matter how many times it is inputted into the perceptron. This contradicts the dynamical behaviour of neurons, where even the injection of a constant input may result in complex patterns described by them. Having a fixed value for the threshold is definitely an option that covers integrate-and-fire-like models (even though the threshold could be dependent on the input), but it does not consider resonator neurons. Over the past few decades, several attempts have been made to address threshold-related issues, including the integration of the Rectified Linear Unit (ReLU) [44] and its variants (DELU [45], GELU [46], SiLU [47] and others), which have proven effective, yielding good results, although the proper spiking mechanism has become obscured as a side effect. However, resonators are still not considered. The parameters being static once training is complete also rules out the possibility of modelling bifurcations. Lastly, in the case of multilayer perceptrons, note that the flow of information is unidirectional (without considering the backpropagation of the error), implying that neurons in future layers can not have an effect on neurons from earlier layers. This scheme is well tied to gradient descent and backpropagation, but it does not account for complex interconnections between neurons.

3.2.2 Convolutional Neural Networks

Frank Rosenblatt developed his framework for a neural model considering neurons that react to optical patterns as stimuli [1]. In order to do so, he considered the spatial arrangement of sensory units and the extraction of features via projection areas, a property that we have omitted until now in order to convey effectively the key ideas behind the perceptron. However, we must now address them in such a way that this suggests that the original perceptron might conceptually be closer to modern convolutional neural networks than to the modern perceptron itself. This is not entirely true, as Rosenblatt apparently did not deliberately think

about the relevance of spatially arranging units and considering hierarchies, but in any case, it is surprising the similarities there are between the original perceptron and convolutional neural networks, suggesting a natural transition between both.

In order to understand the relation between Rosenblatt's perceptron and convolutional neural networks, we need to know that there exists (Rosenblatt describes integrating it as optional) a component between the sensory layer and the association layer known as the projection area, which contains feature extraction units referred to as A_1-units. These units receive input from spatially localised regions of the sensory layer via randomly fixed connections that are confined to those specific regions. In this way, only nearby sensory units influence each processing unit, ensuring that different parts of the sensory region are processed independently before further combining the results. The idea of relying on linear combinations and thresholds is maintained, but unlike in convolutional neural networks, connections are fixed in this design. Still, such a configuration allows sending important visual features to the association layer and filtering out irrelevant signals.

Nevertheless, the detailed understanding of spatial and functional organisation in the visual cortex advanced significantly with the work of David Hubel and Torsten Wiesel, who in 1962 described orientation selectivity and the columnar organisation of neurons in the primary visual cortex [48]. They saw some neurons fire in the presence of bar-like structures in different orientations, a finding that would become the basis for Kunihiko Fukushima's Neocognitron [49], an early hierarchical neural network model for visual recognition. Its architecture is illustrated in Figure 3.4, which somewhat resembles modern convolutional architectures. First of all, layer U_0 works as the retina, which contains sensory input to be processed. All the other layers are combinations of U_{Si} and U_{Ci} modules. Each cell, which is represented inside the tetragons referred to as S-planes in U_{Si} layers (or C-planes in U_{Ci} layers), receives input from a small, localised region of the preceding layer, also known as the **receptive field**. Note that the deeper the layer is, the larger the receptive field of each cell of that layer becomes, as well as that C-planes are connected only to their corresponding S-plane, whereas S-planes receive input from all C-planes. In any case, in order to organise the components, synaptic connections are simulated such that each elliptical region of the preceding layer is integrated by one cell in each S- or C-plane. Note that all the cells in a single cell-plane have receptive fields of the same function, but at different positions. To end with, a key property of the neocognitron is that only S-cells have modifiable connections, since C-cells only aggregate input from their corresponding S-plane. This allows C-cells to fire if a given structure is detected, regardless of its location in a neighbourhood, providing the network pooling-like advantages.

The Neocognitron shares several similarities with modern convolutional neural networks, but its basic neurocomputational properties are still rooted in those of the perceptron. Spatial relations are considered, which is an important feature when dealing with spatially distributed types of data, but the neuronal conceptualisation corresponds to discrete-time integrate-and-fire-like neurons (without a "formal" spike generation mechanism, as in the case of the perceptron). No resonator neurons are considered, time is not modelled once training is over, bifurcations do not play any role in these systems, etc. In any case, we must give them credit

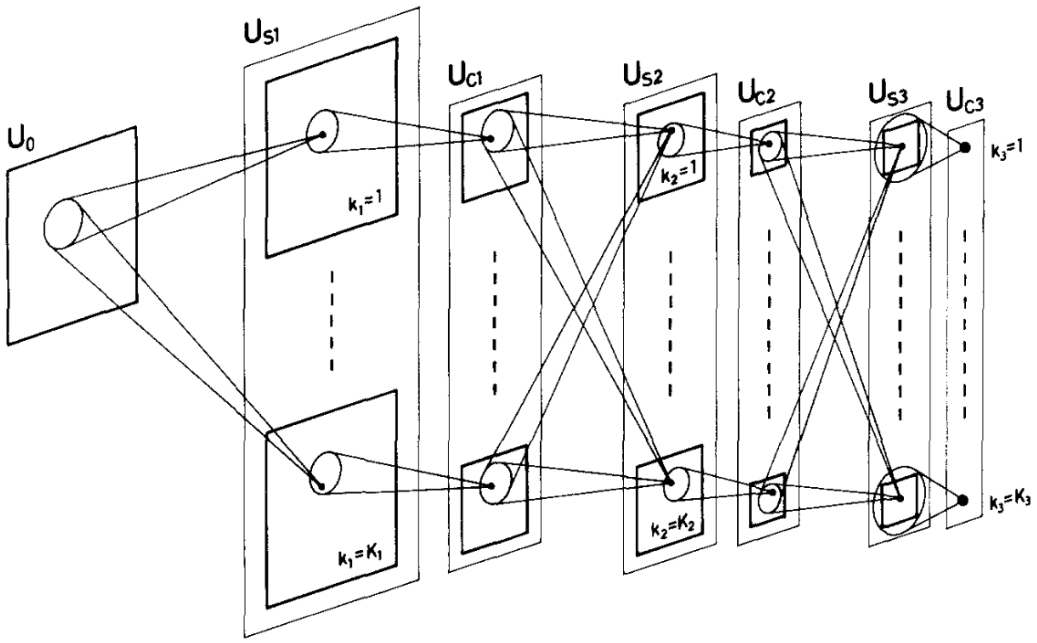

Figure 3.4: Architectural details of the neocognitron. Image taken from [49].

for revolutionising machine learning, and specifically computer vision, with contributions like AlexNet [50] or ResNet [51].

3.2.3 Recurrent Neural Networks

Modelling sequences is, perhaps, the first idea that comes to mind when asked about recurrent neural networks, and they are, perhaps, some of the first neural models that consider time as a relevant factor, although not in the same way as discussed throughout the thesis. Cortical networks in mammalian brains feature densely interlinked feedback pathways that establish recurrent relations between the neurons themselves, and this could be one of the most relevant relations between artificial recurrent networks and biology. In fact, RNNs, and in particular gated-RNNs, such as LSTMs [52], bear a strong resemblance to known cortical structures [53], and that is what will be discussed in this subsection.

On the one hand, artificial RNNs use feedback connections to carry hidden-state information from one time step to the next, directly mirroring cortical recurrence, where neuronal activity propagates through loops over time. Moreover, models mapping inhibitory gating in the cortex to gating mechanisms, as in LSTMs, further reinforce the conceptual alignment between cortical microcircuits and recurrent units. However, unlike in real neurons, where inhibitory signals have a subtractive effect on excitatory signals, the gating mechanisms in artificial networks typically have a multiplicative effect. In any case, this is a relevant biological feature that recurrent architectures consider.

On the other hand, artificial RNNs consider a discrete-time framework to account for the

temporality present in sequential data, aligning this with the hidden state that is updated in different iterations when considering new elements of the sequence. It is true that this allows contemplating dynamical phenomena exhibited by neurons, but note that it is restricted to the integration of new elements rather than to a continuum that is modulated by the elements that are introduced. In other words, time depends on feeding new elements rather than it being an independent factor over which the recurrent architecture is built. Nevertheless, the integration of temporality should be given credit, as recurrent architectures have been successful in complex tasks like language modelling, and they have even been a reasonable basis for other complex architectures, like the Transformer architecture, that also incorporate the notion of time, even if in a discrete manner.

To end with, how about the relationship between recurrent neural networks and computational neuroscience? Aside from their differing conceptualisations of temporality, what are the structural similarities between the two? As with convolutional neural networks and the perceptron, RNNs can be loosely categorised within the integrate-and-fire paradigm, but it must be highlighted that these models do not contain a proper spike generation mechanism either. Their internal mechanism fundamentally relies on linear combinations that integrate inputs from their own outputs (or from other recurrent layers, in architectures where multiple such layers are stacked), but there is no proper spike generation mechanism. Although their hidden states evolve over time, the model parameters remain fixed once training is complete. Therefore, the strengths and limitations discussed in relation to other neural models are also applicable here.

3.2.4 Attention and the Transformer Architecture

While it remains to be seen whether attention is all we need, few would argue against the revolution the Transformer architecture has brought to sequence modelling. Originally published in 2017 [3], the Transformer architecture proposed a truly effective way of modelling sequences by employing a key component known as the attention mechanism, developed by Dzmitry Bahdanau and his collaborators back in 2014 [54]. We live in an era where transformer-based avant-garde applications like ChatGPT are approaching human cognitive abilities, at least in practical terms, so analysing the biological underpinnings of attention could shed light on the capabilities of transformer-based models. This subsection aims to cover such most relevant things.

In a recent study by researchers from MIT and IBM [55], the authors describe how transformers can be constructed from astrocytes and neurons, demonstrating that neuron–astrocyte networks are inherently capable of executing the fundamental operations of the Transformer architecture. This work builds on other studies that establish similarities between the Transformer architecture and hippocampal structures [56], revealing stronger connections between biology and artificial intelligence than previously thought. In fact, the nature of the self-attention mechanism makes it difficult to establish interpretable connections between both fields; the self-attention matrix is calculated via the pairwise dot product between the elements in the sequence, followed by scaling and normalisation operations. These operations are nonlocal in space and time, hence biological interpretations are not trivial. However,

astrocyte-neuron relations may explain the inner mechanism of this architecture.

As a means to show that, the authors propose to build a feedforward perceptron with three layers: input, hidden and output, interspersed with tripartite synapses (which involve a presynaptic neuron, a postsynaptic neuron and an astrocyte process) at the hidden-to-output stage. The system follows a two-stage procedure to implement associative memory. The first stage, known as the **writing phase**, consists of plastically modifying synaptic weights between neurons and between astrocytes to store token feature representations via Hebbian and presynaptic plasticity rules. The role of astrocytes is especially important here, as they integrate the presynaptic activity that each neuron has transmitted to them. The second phase, known as the **reading phase**, consists of performing a fixed forward pass while disabling plasticity updating. Since there are no updates, interactions between the i-th element of a sequence and all the other elements, integrated into the synaptic weights, can be computed via the forward pass. Moreover, the effect of astrocytic processes is also integrated in this phase, resulting in divisive modulations equivalent to the softmax normalisation. Therefore, we see that transformers can be tightly linked to associative memory and neuron-astrocyte network configurations.

And what about the biological implications of everything seen in this subsection? While it is not that easy to establish and justify similarities between the Transformer architecture and neurocomputational models due to the abstract conceptualisation of this modelling scheme, we can comment on several characteristics. First of all, by employing known astrocyte dynamics (calcium-mediated spatial averaging and modulatory feedback), the model offers an experimentally testable account of how the brain could implement attention-like operations. In addition to this, we could justify the closer relation between transformers and integrate-and-fire-like models than between them and resonate-and-fire-like models, as transformers perform weighted summations (integrations), rather than exhibit oscillatory, frequency-tuned behavioural patterns of resonate-and-fire-like models. Therefore, we could include all the strengths and weaknesses commented on in previous sections here, especially those related to the invariable nature of the threshold and the time-independency of the parameters of the model once training is over, but in this case it is not that trivial to bridge the gap between computational neuroscience and the abstract conceptualisation of the brain given by the Transformer architecture.

This concludes this section, where some of the most important connections between computational neuroscience, neural-based machine learning and deep learning have been provided to understand how artificial intelligence may be built upon biological plausibility assumptions. The examination of the strengths and weaknesses of current modelling paradigms will serve as the primary motivation for the material covered in the following chapter.

CHAPTER 4

Theoretical Framework for a Biologically Plausible Neural Model and its Applicability in Artificial Intelligence

After reviewing some of the most fundamental approaches towards modelling neurons, highlighting both their strengths and downsides, this chapter focuses on developing a neural model that integrates the fundamental principles and mechanisms of computational neuroscience into a neural-based machine-learning-compatible framework. As noted on several occasions, although deep learning has revolutionised artificial intelligence by emulating neuronal information transmission and processing, it still lacks some of the fundamental neural mechanisms that enable richer information flow and more abstract representations of reality. Operating under the premise that more biologically plausible neural models can yield more efficient and effective solutions, this thesis proposes incorporating some of these missing mechanisms and comparing the outcomes with those produced by the simplest neural architectures used in machine learning. It could surely be the case that simpler formulations actually perform better, but exploring new alternatives may be the key to building robust artificial intelligence applications.

This chapter will cover the fundamental aspects of a biologically plausible artificial neural model that we have devised by integrating some of the theoretical components developed in Chapters 2 and 3 (at least, a model that tries to capture some phenomena that current architectures miss). Surely, there exist many modelling attempts that integrate biologically plausible components into neural models, such as spiking neural networks [57]. Nevertheless, some of those models deviate from the core idea of machine learning and require specialised knowledge about how neurons process data, among others. That is the reason for which we

have tried to integrate some components we consider as fundamental towards biological plausibility, trying not to deviate from current existing neural-based machine learning frameworks. We consider that a soft transition between computational neuroscience and machine learning is a positive feature, even more considering that there are few works that propose developing artificial intelligence under that premise [43]. Therefore, we will motivate the need for such a model by analysing which the most relevant shortcomings of current neural-based machine learning and deep learning approaches are, characterising the model formally and analysing its viability, biological plausibility and the extent to which it can be integrated into current machine learning frameworks. Developing and characterising a neural model is not a trivial task, and many decisions have to be made to devise a robust and consistent formulation that satisfies our requirements.

4.1 Motivating the Need for a More Comprehensive and Complete Neural Model

As mentioned earlier, deep learning approaches nowadays offer robust solutions for building artificial-intelligence–related applications, but they are, to a certain extent, devoid of biological plausibility. On the one hand, current artificial neurons model reasonably well integrate-and-fire-like neurons on a static framework, but they scarcely address the dynamism of some internal states that should, in theory, determine the neuron's time-dependent behaviour. Recurrent neural networks and spiking neural networks can be thought of as the closest approach to managing time-dependent factors, but their formulation is typically tied to handling sequential data or biologically meaningful signals rather than to describing an input-agnostic system. In general, time is typically not considered a relevant factor when building neural networks. On the other hand, resonator-like neurons still need a greater presence in neural models. And what about efficiency and effectiveness? In recent years, state-of-the-art models have proven capable of solving many daunting tasks for which they were conceived, but efficiently addressing those challenges remains an open question.

In summary, the biological plausibility in modern artificial intelligence models is quite limited. We need to explore some other alternatives that cover the previously commented shortcomings (without discarding the neurocomputational properties that are currently being modelled correctly). In fact, our cognition and abstract thinking come from very complex neural phenomena, some of which we still do not comprehend. Therefore, exploring alternatives that try to include these characteristics could be a determining factor for modelling abstraction and cognition. A neural model possessing internal time-dependent states that are modulated by input signals, when properly trained, may exhibit complex behaviour related to cognition, which is our ultimate goal. There is surely no guarantee that such a model will outperform everything we have built so far, nor that it will be more efficient than current solutions, but that is precisely what we need to verify. Should we obtain meaningful results, expanding the scope of artificial intelligence to embrace this new paradigm could prove advantageous to all. That is the reason why this thesis has been developed. We shall discuss now the formalisation that characterises such a biologically more plausible and complete neural model.

4.2 Characterisation of our Proposal for a Biologically Plausible Neural Model

This thesis will now cover how classic computational neuroscience and neural-based machine learning can intersect into a mixed paradigm that tries to integrate some of the most interesting aspects of each field. We have mentioned on many occasions how time and internal states are essential aspects in neural models. As a consequence, this section will cover the theoretical background for a neural model that integrates those two components into an existing machine learning formalisation. This idea was first introduced in 2018 through a new family of deep neural network models based on ordinary differential equations [43]. Instead of relying on discrete sequences of hidden layers, this work proposes parameterising the derivative of the hidden state to represent the neural network. We will not directly represent a neural network through the derivative of a hidden state, but insert the derivative of an internal state into a single neuron, much as in spiking neural networks [57]. However, instead of relying on spiking neural networks, where discrete spike events are modelled, we propose developing a continuous framework, much like in neural ordinary differential equations. This is, we shall characterise a neural model by including some aspects of neural-based machine learning that we consider as relevant, neural ordinary differential equations and spiking neural networks, developing a continuous-time spiking neural network. This is a relatively new, active and promising research field [58, 59] that builds on existing artificial intelligence architectures by including biologically plausible neural features.

4.2.1 Characterisation of the Architecture

The architecture we propose developing is represented in Figure 4.1. It shows an integrate-and-fire-like formulation in which a continuous input, represented as a series of features (a series of functions), is integrated via a linear combination to compute the injected current that will modify the internal state of the neuron. We will restrict ourselves to one-dimensional models by employing a quadratic integrate-and-fire formulation for the internal state, membrane voltage. Although one-dimensional models lack the necessary capacity to model Andronov-Hopf bifurcations, typical in resonator neurons (see Sections 2.3.2.1 and 3.1.3), we have mentioned that the quadratic integrate-and-fire model is a particularly interesting one-dimensional model; apart from being a canonical model, as it is the topological normal form for the saddle-node bifurcation and thus it can represent a wide array of neural models, it is a genuine spiking model. Unlike the leaky integrate-and-fire model, typically employed in spiking neural networks and neural ordinary differential equations, the quadratic integrate-and-fire model inherently models natural spikes. That is the reason for which this is a suitable and robust parameterisation for the internal state of the neuron.

As for the internal state of the neuron, there are some important considerations to be made before addressing how such a model can be trained. First of all, we expand the quadratic integrate-and-fire formulation, given in Equation 3.2, by including the current to be injected. This does not have a particularly relevant impact on the formulation, as for a constant input we can define $b' = b + I_{\text{inj}}$. Varying either the input or the value of b over time corresponds to

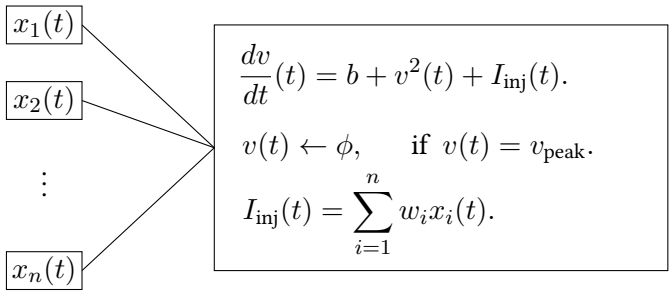

Figure 4.1: Diagram of the flow of information in the biologically plausible neural model. While this diagram incorporates many features common to neural-based machine learning approaches, its use of a quadratic integrate-and-fire model for the neuron's continuous internal state provides a biologically plausible framework for developing artificial intelligence.

shifting the function that represents the time derivative of the membrane voltage, as analysed in Figure 3.2. Therefore, the formulation corresponds to a quadratic integrate-and-fire model where $b' = b + I_{\text{inj}}$. Note that under those circumstances, we can infer a basic property like the value for the threshold: if $b' < 0$, then $v_{\text{thresh}} = \sqrt{|b'|} = \sqrt{|b + I_{\text{inj}}|}$; if $\phi > b'$, then the model will enter a never-ending spiking state.

On the other hand, a series of learnable parameters has been included in the formulation so that this model is trainable by employing machine learning techniques. The learnable parameters include the coefficients w_i for the integration of the input and the voltage reset value ϕ. In contrast, the peak value that will determine when the voltage returns to its reset value is left as a hyperparameter. As far as we are concerned, no continuous-time spiking neural network offers the exact configuration we propose (in addition to other components we will introduce later on) as of today, so we think it is fundamental that we analyse how a learnable reset value will affect the model's capacity and effectiveness before including both a learnable peak and reset value for voltage. Therefore, v_{peak} will be a hyperparameter.

Finally, an important consideration that will be relevant when developing the theoretical training framework is that assigning $v(t)$ the value of ϕ when $v(t) = v_{\text{peak}}$ essentially means that $v(t)$ will be equal to v_{peak} in the infinitesimal instant before the jump occurs but that its value will be equal to the reset value. This is, $v(t)$ will be a right-continuous function such that for every spiking event at time step t_k the conditions represented in Equation (4.1) hold:

$$\begin{cases} v(t_k^-) = \lim\limits_{t \to t_k^-} v(t) = v_{\text{peak}}. \\ v(t_k^+) = \lim\limits_{t \to t_k^+} v(t) = \phi = v(t_k). \end{cases} \tag{4.1}$$

Once the general structure has been introduced, we shall now develop other theoretical concerns regarding the training of the model. Since we are truly interested in integrating this characterisation into the machine learning framework, we find it useful to employ the toolset it offers to train models. Backpropagation [60], a core component of neural network training, offers an intuitive and efficient way to train large networks so that they can adapt to a given

task. We are aware that there is widespread criticism of the actual biological plausibility of backpropagation [61, 62], but, as noted earlier, we need to strike a balance between biological plausibility, computational efficiency and ease of integration. Future research could explore biologically plausible training alternatives. In any case, combining spiking neural networks with neural ordinary differential equations, we ensure a higher level of biological plausibility in our neural model. On the other hand, by considering backpropagation, we ensure a computationally efficient training scheme and the direct integration of the neural model into the machine learning framework. We find that this is an acceptable balance between all the aforementioned components. As a consequence, now that we have formally characterised our model, we shall discuss how training can be performed by considering the typical forward and backward passes of a backpropagation-based training step.

4.2.2 Forward Flow of Information

The first training step consists of computing the output of the model and comparing it with some target values to analyse whether the model is performing as expected or whether it needs more training.

4.2.2.1 Input Representation

Unlike in classic neural-based machine learning approaches, where the forward pass consists of a discrete step that integrates the input features, we now have continuous features that represent the effect of each feature on a given time step. Moreover, those features enable the neuron's internal state to evolve as its ordinary differential equation dictates. Therefore, first of all, we need to explicitly define the structure of the input features.

When creating datasets, instances contain values that represent their inner structure; these are known as features. Images are matrices that store intensity values for each primary colour, physiological data contain health indicators for different patients, etc. There is no natural relation between what the data represents and time, so we need to transform the data in such a way that a feature vector $(x_1, \ldots, x_n)$ can be represented as a vector of functions $(x_1(t), \ldots, x_n(t))$. Our approach proposes the following transformation strategies:

- **Zero-Order Hold Model.** We consider this to be the most intuitive and simple approach as to how inputs stimulate neurons. Given a feature vector $(x_1, \ldots, x_n)$ and after defining a time window $[0, T]$ for the stimulation period, its time-dependent representation will be defined as Equation (4.2) shows:

$$(x_1(t), \ldots, x_n(t)) = \begin{cases} (x_1, \ldots, x_n), & \text{if } t \in [t_1, t_2] \subseteq [0, T], \\ (0, \ldots, 0) & \text{else,} \end{cases} \tag{4.2}$$

  where t_1 and t_2 represent the limits of the input injection time range. In other words, this will be an n-dimensional rectangular signal; for time steps outside the injection range, $[0, t_1) \cup (t_2, T]$, the neuron will receive no input at all (after calculating the linear combination, we have that $(w_1, \ldots, w_n) \cdot (0, \ldots, 0) = 0$), but for all the time steps in

the injection range $[t_1, t_2]$, the neuron will receive a constant input. It is reasonable to argue that for a sufficiently long stimulation and injection period, the neuron will have had enough time to either fire or not (or, at least, to fire either more or less frequently). Naturally, we will need to find good candidates for t_1, t_2 and T, but that need not concern us yet.

- **Dynamical Source of Input.** Since reality is continuous and volatile, neurons receive inputs that practically change from each time step to another, even if those inputs refer to the same being. For example, photoreceptors are continuously exposed to photons, which stimulate them and enable us to perceive the world visually, and even if they receive inputs in a time step $t_l + \epsilon$ under the same conditions (the same landscape, the same portrait, etc.), it is highly unlikely that the neural input will be exactly identical. It could be the case that the photon collided with the photoreceptor at a slightly different angle in time step $t_l + \epsilon$ than the photon in t_l, even if they contain the same amount of energy and other properties, contributing to a slight modulation of the injected current.

  That is the reason why we propose the dynamical source of input technique. The idea is similar to zero-order hold in that we will have a stimulation interval $[0, T]$, an injection interval $[t_1, t_2] \subseteq [0, T]$, but instead of injecting constant inputs, we will inject slightly modulated inputs on contiguous subintervals of $[t_1, t_2]$. Let S denote the number of subintervals in which the input will be modulated and $S_i \subseteq [t_1, t_2]$ the i-th subinterval ($0 \leq i \leq S - 1$). Then, the injected input will be calculated as Equation (4.3) shows:

$$I_{\text{inj}}(t \in S_i) = (f_1^{(i)}(x_1), \ldots, f_n^{(i)}(x_n)). \tag{4.3}$$

  $f_j^{(i)}$ refers to a transformation applied to the j-th feature at the subinterval S_i. We believe that this allows the neuron to incorporate these time-variant inputs into its rich dynamics, allowing it to solve more complex problems.

4.2.2.2 Spikes as the Primary Representation of Neural Response

We know how the internal state of the neuron evolves and how inputs will be fed to the neuron, but we are missing an important component: the representation of spikes. One may think that spikes can be extracted after computing $v(t)$ for every time step in $[0, T]$, but there are some concerns that need to be addressed with regard to this. Suppose that $s(t)$, the function that determines the time steps in which the membrane voltage has reached the value v_{peak}, is computed employing the following expression:

$$s(t) = \begin{cases} 1, & \text{if } v(t) - v_{\text{peak}} \geq 0. \\ 0, & \text{else.} \end{cases}$$

However, this expression is actually inappropriate, since $v(t)$ is right-continuous; there will be no value for t where $v(t) - v_{\text{peak}} \geq 0$, and therefore, $s(t) = 0,\ \forall t \in [0, T]$. We need some other alternative that is both coherent and simple to compute given $v(t)$. A typical strategy consists of working with soft readouts, where $s(t)$ is calculated as Equation (4.4) shows:

$$s(t) = f(v(t)), \tag{4.4}$$

where f is a smooth function, typically a sigmoid-like function (we shall introduce all the results for $f(v(t)) = \sigma(v(t)) - 0.5$, but other options are perfectly valid). At this point, it is important to note that $v(t)$, and therefore, $s(t)$, are not necessarily continuous with respect to time, but they are with respect to the learnable parameters. This will be an important consideration when computing the gradient of the loss function with respect to the learnable parameters. Moreover, even if $v(t)$ is not continuous with respect to time, it will be bounded, and the set $\{t_1, \ldots, t_K\} \subset [0, T]$, which contains the time steps where $v(t)$ is not continuous, will have a Lebesgue measure of 0. This implies that both $v(t)$ and $s(t)$ will be Riemann integrable, a property we will need afterward.

4.2.2.3 Output of the Model and Loss Function

The last step in the forward pass consists of obtaining the output of the model and calculating a loss considering that output. In our case, the loss will be calculated on top of $s(t)$, as it is the signal that best reflects whether the neuron has responded intensively to a given input or whether the input has caused minor spiking activity. If $s(t)$ contains many peaks, that may indicate that the neuron is responding to that given input positively, whereas lack of activity may indicate that the input being integrated has either no major effect on the dynamics or that it is driving the neuron to its resting state more frequently. If we train the neuron in such a manner that, for example, instances belonging to a class cause major spiking activity and instances belonging to the other class drive the neuron to its resting state, we will have effectively trained the neuron to fire at specifically structured inputs. This reflects, to some extent, the way our neurons respond to stimuli, which represents a relevant modelling feature.

However, how can we quantify if the neuron's spiking activity is greater in some circumstances than in others, and how can we calculate the actual output of the model? Should we sum the number of times $v(t)$ reaches its maximum value? Should we take the spiking frequency? Should we also consider the effect of membrane voltage decreasing to negative values? An option that combines many of these characteristics consists of integrating $s(t)$ over the stimulation domain. This is, given the signal $s(t)$, we will integrate it from 0 to T. Finally, calculating the output requires determining the nature of the task being solved. Since we are formalising the structure of a single neuron, we shall tackle the simplest learning challenge: binary classification. Other types of tasks will probably require building neural networks or complex architectures, but we will not cover that in this thesis. Therefore, being in a binary classification scenario, the output of the model will be calculated by applying a sigmoid function over the previous integral (we will scale the argument so that the result of the integral can be extended to span a greater probabilistic space); loss will be computed on top of that. In summary, Equation (4.5) shows how the output of the model can be calculated from the signal $s(t)$.

$$u(s) = u(v) = \int_0^T s(t)dt = \int_0^T [\sigma(v(t)) - 0.5]dt \Longrightarrow \hat{y}(u) = \sigma(\kappa u). \tag{4.5}$$

Note that $u(s)$ (or $u(v)$) is a scalar functional; it takes a function as input and returns a scalar. This will be important when deriving the gradient.

Finally, how about the loss function? What is it that we aim to optimise? Now that we have calculated the output of the model, it is safe to say that we can use any loss function that is employed in binary classification. We do not need to make any further modifications to adapt the output to the loss. Therefore, in order to provide an explicit formula for the gradient, we will take the binary cross-entropy loss as our reference loss function, as shown in Equation (4.6), but other functions can also be applied with minor modifications. In fact, other optimisation goals could be addressed, like early classification or speed on prediction, but we will cover the most basic case here.

$$\mathcal{L}(y, \hat{y}) = -[y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})]. \tag{4.6}$$

This concludes the forward pass of a training step, but we now have to optimise the learnable parameters in order to minimise the value of $\mathcal{L}$ for the instances in a given dataset.

4.2.3 Optimisation of Learnable Parameters I: Analytical Gradient

We shall now discuss how parameters can be iteratively optimised by employing gradient descent. As in classic neural-based machine learning approaches, where the principal motivation lies in that the gradient of the loss with respect to the trainable parameters provides the direction with the maximum positive slope, and therefore, following the opposite direction drives the loss towards a local optimum, we shall follow the same philosophy. This first subsection will cover how the gradient can be analytically calculated, providing closed formulas for the gradient of the loss with respect to the trainable parameters. However, we will see that computing the partial derivatives by employing those formulas is not computationally efficient (both in time and memory). As a consequence, the second subsection will introduce a memory-efficient approach to compute the gradient of the loss.

Remark. While computing the partial derivatives, we will employ different parameterisations for some functions. This is because we will be interested in highlighting some specific components in some cases. For example, the loss is a function of $\hat{y}$, but $\hat{y}$ itself, apart from being a variable, is also a function of $v(t)$. Therefore, as long as the notation is consistent, we can refer to $\mathcal{L}$ as a function of $\hat{y}$, as a function of $s(t)$, as a function of $v(t)$, as a function of θ (the set of learnable parameters) or $w_1, \ldots, w_n, b, \phi$, etc.

Let us begin by deriving the expression for the partial derivative of $\mathcal{L}$ with respect to the integration weights w_i. By applying the derivative of the composition of functions, also known as the chain rule, we can decompose our target partial derivative as follows:

$$\frac{\partial \mathcal{L}}{\partial w_i}(y, \hat{y}) = \frac{d\mathcal{L}}{d\hat{y}}(\hat{y}) \cdot \frac{d\hat{y}}{du}(u) \cdot \frac{du}{dv}(v) \cdot \frac{\partial v}{\partial w_i}(w_i).$$

We shall derive the analytical expression for each term:

- As for the derivative of the loss with respect to the output of the model, the derivation corresponds to differentiating the binary cross-entropy function taking $\hat{y}$ as a variable.

Therefore,

$$\frac{d\mathcal{L}}{d\hat{y}}(\hat{y}) = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} = \frac{-y\,(1-\hat{y}) + \hat{y}\,(1-y)}{\hat{y}\,(1-\hat{y})} = \frac{-y + y\hat{y} + \hat{y} - \hat{y}y}{\hat{y}\,(1-\hat{y})}$$
$$= \frac{\hat{y}-y}{\hat{y}\,(1-\hat{y})} = \frac{\hat{y}-y}{\sigma(\kappa u)\,(1-\sigma(\kappa u))} = \frac{\hat{y}-y}{\sigma'(\kappa u)}.$$

Note that $\hat{y} = \hat{y}(u) = \sigma(\kappa u)$ and that $\sigma'(\kappa u) = \sigma(\kappa u)\,(1-\sigma(\kappa u))$. These equalities will help simplify the expression later.

- As for the derivative of the output of the model with respect to the variable u, the result corresponds to the derivative of the sigmoid function multiplied by the gain factor. Therefore,

$$\frac{d\hat{y}}{du}(u) = \kappa\sigma'(\kappa u).$$

- As for the derivative of the scalar functional u with respect to the trajectory $v(t)$, the notion of the derivative must be extended to include these types of functions. Before delving into that, let us analyse which the derivative that we actually need to compute is:

$$\frac{du}{dv}(v) = \frac{d}{dv}\left[\int_0^T s(t)dt\right](v) = \frac{d}{dv}\left[\int_0^T [\sigma(v(t)) - 0.5]dt\right](v).$$

The main challenge arises when trying to develop the previous expression, as we need some abstract tools to handle the derivatives of scalar functionals. It is the case that Gateaux derivatives [63] are employed in this situation, as they are the generalisation of directional derivatives. As with functions of many variables, where directional derivatives indicate the slope of the function in a given direction, expressed by a vector, Gateaux derivatives allow us to generalise the notion of directional derivatives without moving away from that philosophy. The core idea remains the same, but instead of using vectors as directions, trajectories or functions will fulfill that role. Given a trajectory $\eta(t)$, the Gateaux derivative of a scalar functional $u(v(t))$ in the direction η is defined in Equation (4.7):

$$\delta u[v](\eta) = \lim_{\epsilon \to 0} \frac{u(v+\epsilon\eta) - u(v)}{\epsilon}. \tag{4.7}$$

As with partial derivatives of functions of many variables, where the vectors corresponding to the canonical basis are chosen as the directions for the derivatives, a similar idea can be applied to Gateaux derivatives, leading to what is known as the **functional derivative** of a scalar functional [64]. By definition, if the Gateaux derivative of the scalar functional u with respect to any trajectory can be written as Equation (4.8) shows, then we will say that the functional derivative of u exists.

$$\delta u[v](\eta) = \int_0^T \frac{\delta u}{\delta v}(t)\eta(t)dt. \tag{4.8}$$

The expression $\frac{\delta u}{\delta v}(t)$ corresponds to a function of t, and that function is referred to as the functional derivative of u at v. If such an expression exists, then

$$\frac{du}{dv}(v) = \int_0^T \frac{\delta u}{\delta v}(t)dt.$$

Following Equation (4.7), let us see whether we can derive such an expression in our case. By considering any trajectory $\eta(t)$, the Gateaux derivative of $u(v(t))$ in the direction η can be calculated as follows:

$$\begin{aligned}\delta u[v](\eta) &= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\left(\int_0^T[\sigma(v(t)+\epsilon\eta(t))-0.5]dt - \int_0^T[\sigma(v(t))-0.5]dt\right)\\ &= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\int_0^T[\sigma(v(t)+\epsilon\eta(t))-\sigma(v(t))]dt.\end{aligned}$$

Note that

$$\begin{aligned}\lim_{\epsilon\to 0}\frac{1}{\epsilon}[\sigma(v(t)+\epsilon\eta(t))-\sigma(v(t))] &\overset{\begin{bmatrix}x=v(t)\\ h=\epsilon\eta(t)\end{bmatrix}}{=} \lim_{h\to 0}\frac{1}{\frac{h}{\eta(t)}}[\sigma(x+\epsilon)-\sigma(x)]\\ &= \sigma'(x)\eta(t) = \sigma'(v(t))\eta(t).\end{aligned}$$

Now, since $\forall\epsilon > 0$ the integrand is Riemann integrable (supposing that η is Riemann integrable), the limit of the integrand as $\epsilon \to 0$ exists and that limit is bounded by $||\sigma'||_\infty\,|\eta(t)|$, we can interchange the limit and the integral, resulting in that

$$\delta u[v](\eta) = \int_0^T \sigma'(v(t))\eta(t)dt.$$

This is precisely the expression we were looking for, as we immediately infer that $\frac{\delta u}{\delta v}(t) = \sigma'(v(t))$. Therefore,

$$\frac{du}{dv}(v) = \int_0^T \sigma'(v(t))dt.$$

- As for the partial derivatives of the voltage trajectory with respect to the integration coefficients w_i (as well as the parameters b and ϕ), we will find a closed formula for them by operating on the ordinary differential equation that characterises our neural model. Recall that

$$\frac{dv}{dt}(t) = b + v^2(t) + \sum_{i=1}^n w_i x_i(t).$$

Therefore, differentiating the derivative function with respect to w_i yields

$$\frac{\partial}{\partial w_i}\left[\frac{dv}{dt}\right](t, w_i) = \frac{\partial}{\partial w_i}\left[b + v^2(t) + \sum_{i=1}^n w_i x_i(t)\right](t, w_i).$$

However, computing that derivative is not that trivial, since v is a function that depends on the learnable parameters. As a consequence, instead of tackling that problem, we shall interchange the derivatives. Since smoothness is preserved by all the variables in the expression above (by only considering the differential equation, which is the case we are developing now, v will be differentiable in all its domain), interchanging derivatives should have no effect on the result. Therefore,

$$\begin{aligned}\frac{\partial}{\partial w_i}\left[\frac{dv}{dt}\right](t, w_i) = \frac{d}{dt}\left[\frac{\partial v}{\partial w_i}\right](t, w_i) &= \frac{\partial}{\partial w_i}[v^2(t)](t, w_i) + x_i(t) \\ &= 2v(t)\frac{\partial v}{\partial w_i}(t, w_i) + x_i(t).\end{aligned}$$

Our new objective will be to solve an ordinary differential equation that will allow us to infer the result we are looking for. By referring to $\frac{\partial v}{\partial w_i}(t, w_i)$ as $p_{w_i}(t)$, note that

$$\frac{dp_{w_i}}{dt}(t) = 2v(t)p_{w_i}(t) + x_i(t).$$

The differential equation will be solved using the integrating factor technique as follows. This requires multiplying a function $\mu : \mathbb{R} \to \mathbb{R}$ on both sides of the equality. μ takes the following form:

$$\mu(t) = \exp\left(-\int_0^t 2v(s)ds\right).$$

Therefore,

$$\begin{aligned}&\frac{dp_{w_i}}{dt}(t) - 2v(t)p_{w_i}(t) = x_i(t) \implies \\ &\mu(t)\frac{dp_{w_i}}{dt}(t) - 2\mu(t)v(t)p_{w_i}(t) = \mu(t)x_i(t).\end{aligned}$$

In such a scenario, note that

$$\begin{aligned}\frac{d}{dt}\left[\mu(t)p_{w_i}(t)\right](t) &= \mu(t)\frac{dp_{w_i}}{dt}(t) + \frac{d\mu}{dt}(t)p_{w_i}(t) \\ &= \mu(t)\frac{dp_{w_i}}{dt}(t) + \frac{d}{dt}\left[\exp\left(-\int_0^t 2v(s)ds\right)\right](t)\, p_{w_i}(t) \\ &= \mu(t)\frac{dp_{w_i}}{dt}(t) - \mu(t)\frac{d}{dt}\left[\int_0^t 2v(s)ds\right](t)\, p_{w_i}(t).\end{aligned}$$

By the fundamental theorem of calculus, the previous expression can be reduced to

$$\frac{d}{dt}\left[\mu(t)p_{w_i}(t)\right](t) = \mu(t)\frac{dp_{w_i}}{dt}(t) - 2\mu(t)v(t)p_{w_i}(t).$$

That is exactly the expression we had in the left side of the equality when multiplying $\mu(t)$ to the differential equation (after introducing the integrating factor). Therefore,

$$\frac{d}{dt}\left[\mu(t)p_{w_i}(t)\right](t) = \mu(t)x_i(t).$$

Finally, by integrating both sides, we have that

$$\int_0^t \frac{d}{ds}[\mu(s)p_{w_i}(s)](s)\, ds = \int_0^t \mu(s)x_i(s)ds.$$

Applying the second fundamental theorem of calculus and taking $p_{w_i}(0) = 0$, since membrane voltage is not dependent on w_i at $t = 0$ (it is set to ϕ), the previous expression can be equivalently written as

$$\mu(t)p_{w_i}(t) - \mu(0)p_{w_i}(0) = \int_0^t \mu(s)x_i(s)ds \implies p_{w_i}(t) = \frac{1}{\mu(t)}\int_0^t \mu(s)x_i(s)ds.$$

Or equivalently,

$$p_{w_i}(t) = \exp\left(\int_0^t 2v(s)ds\right)\int_0^t \exp\left(-\int_0^s 2v(r)dr\right)x_i(s)ds.$$

Remark. The only aspect that we need to consider is that our trajectories will not always follow the differential equation, as there will be some time steps $\{t_1, \ldots, t_K\}$ where the membrane voltage will be set to its resting state. However, this is only a minor inconvenience, as we can apply this strategy after arranging $v(t)$ in intervals (t_k, t_{k+1}), where each interval will contain a subtrajectory that follows the ordinary differential equation of our neural model.

Now that we have successfully calculated all the components needed to apply the chain rule, we only need to combine them (and consider the intervals where $v(t)$ is continuous) to obtain the partial derivative of the loss with respect to w_i. Equation (4.9) represents the aggregated result.

$$\frac{\partial \mathcal{L}}{\partial w_i}(w_1, \ldots, w_n, b, \phi) = \sum_{k=0}^{K-1}\int_{t_k}^{t_{k+1}} \kappa(\hat{y} - y)\sigma'(v(t))\left[\exp\left(\int_{t_k}^t 2v(s)ds\right)\int_{t_k}^t \exp\left(-\int_{t_k}^s 2v(r)dr\right)x_i(s)ds\right]dt. \quad (4.9)$$

As for the partial derivative of the loss with respect to b, the derivation is almost identical, with the exception that when calculating the partial derivative of v with respect to b, the differential equation we will need to solve will be the following one:

$$\frac{dp_b}{dt}(t) = 1 + 2v(t)p_b(t).$$

In any case, the integrating factor is identically employed, resulting in an expression that bears a strong resemblance to the one shown in Equation (4.9). In this case, Equation (4.10) shows the analytical expression for the partial derivative of the loss with respect to b.

$$\frac{\partial \mathcal{L}}{\partial b}(w_1, \ldots, w_n, b, \phi) = \sum_{k=0}^{K-1}\int_{t_k}^{t_{k+1}} \kappa(\hat{y} - y)\sigma'(v(t))\left[\exp\left(\int_{t_k}^t 2v(s)ds\right)\int_{t_k}^t \exp\left(-\int_{t_k}^s 2v(r)dr\right)ds\right]dt. \quad (4.10)$$

And what about the partial derivative of the loss with respect to ϕ? As in the other two cases, applying the chain rule results in three identical terms and a fourth term that contains

the partial derivative of v with respect to the learnable parameter. By considering any interval (t_k, t_{k+1}), we know that $v(t)$ will follow the differential equation corresponding to the neural model. This results in that we will need to solve the following differential equation to obtain the partial derivative we aim to calculate:

$$\frac{dp_\phi}{dt}(t) = 2v(t)p_\phi(t).$$

Solving the differential equation by employing the integrating factor yields

$$p_\phi(t) = \exp\left(\int_{t_k}^{t} 2v(s)ds\right),\ t \in (t_k, t_{k+1}).$$

Taking into account the previous result, the partial derivative of the loss with respect to ϕ is shown in Equation (4.11).

$$\frac{\partial \mathcal{L}}{\partial \phi}(w_1, \ldots, w_n, b, \phi) = \sum_{k=0}^{K-1} \int_{t_k}^{t_{k+1}} \kappa(\hat{y} - y)\sigma'(v(t)) \exp\left(\int_{t_k}^{t} 2v(s)ds\right) dt. \quad (4.11)$$

4.2.4 Optimisation of Learnable Parameters II: Adjoint Method

Although we have successfully derived the gradient of the loss with respect to the learnable parameters, the methodology employed nonetheless exhibits certain shortcomings that render the approach computationally inefficient. To begin with, note that we have had to solve an ordinary differential equation per learnable parameter. It is true that in this specific case, the solution can be calculated analytically, and therefore, no numerical integration method is required. However, for neural models that involve multiple state variables, the characteristic differential equation may no longer be analytically tractable. As a consequence, calculating each partial derivative would require solving a differential equation numerically, resulting in a complexity of, at least, $\mathcal{O}(mp)$, where m refers to the number of state variables and p the number of learnable parameters. Moreover, in order to compute the gradient of the loss, we will have to store the full Jacobian trajectory, which is a $n \times p$ matrix.

In addition, even in this case, where we have managed to derive closed formulas for the partial derivatives, note that calculating $\frac{\partial \mathcal{L}}{\partial w_i}(w_1, \ldots, w_n, b, \phi)$ and $\frac{\partial \mathcal{L}}{\partial b}(w_1, \ldots, w_n, b, \phi)$ requires solving nested integrals. During training our model using computational resources, we will not be able to calculate the whole $v(t)$ function, since we will have to build $v(t)$ by solving the differential equation numerically and considering the transition steps (even if $v(t)$ can be analytically calculated, we cannot store an infinite number of values of $v(t)$ corresponding to the interval $[0, T]$ on a computer). Therefore, the integrals will have to be solved numerically, and in order to obtain a precise estimation of the actual result, the solver will have to consider a large number of subintervals in $[0, T]$. If we denote the number of subintervals as N, solving the nested integrals requires a computational cost of, at least, $\mathcal{O}(N^2)$. This is definitely an undesirable situation.

Since the previous two issues pose a daunting computational challenge with regard to efficiency, instead, we will develop an alternative to obtain a memory-efficient solution to

compute the gradient of the loss with respect to the learnable parameters. The alternative solution revolves around the **adjoint state method** [59, 65], which refers to solving the optimisation problem (in our case, minimising the loss) via the Lagrangian technique. The procedure, although mathematically exhaustive, can be intuitively understood if the reader is familiar with the Lagrangian method. Let us address the full derivation step by step.

Recall that we can reformulate our optimisation goal as minimising the loss function $\mathcal{L}(v;\theta)$ over all possible trajectories $v : [0,T] \to \mathbb{R}$ and learnable parameters θ (in our case, $w_1, \ldots, w_n$, b and ϕ). However, the optimisation procedure must be subject to the constraint determined by the differential equation of the neural model, i.e., each solution $v(t)$ must satisfy

$$\frac{dv}{dt}(t) = b + v^2(t) + \sum_{i=1}^{n} w_i x_i(t).$$

We also have a second constraint related to spiking transitions. Each $v(t)$ must also satisfy that for each t_k corresponding to a spike,

$$v(t_k) = v(t_k^+) = \phi$$

holds. In such a scenario, the Lagrangian method consists of operating on a function that introduces the constraints as additive terms multiplied by some terms referred to as multipliers. In our case, the scenario is somehow more complex, as one of the constraints that needs to be fulfilled is a differential constraint. This implies that one of the Lagrange multipliers will be a function that will be multiplied to the differential constraint and integrated in the interval $[0,T]$. Note also that the Lagrangian function will be a functional, as it takes a function as input. Under these circumstances, Equation (4.12) shows the Lagrangian function corresponding to our minimisation problem.

$$\mathcal{J}(v,\lambda,\mu_1,\ldots,\mu_K;\theta) = \mathcal{L}(v;\theta) + \int_0^T \lambda(t)\left[\frac{dv}{dt}(t) - \left(b + v^2(t) + \sum_{i=1}^{n} w_i x_i(t)\right)\right] dt + \sum_{k=1}^{K} \mu_k(v(t_k^+) - \phi). \quad (4.12)$$

We shall be referring to $b + v^2(t) + \sum_{i=1}^{n} w_i x_i(t)$ as $f(v(t);\theta)$ from now on.

So, how can we employ that function to extract an alternative expression for the partial derivatives? In theory, the Lagrangian is used to calculate a tuple $(v^*, \lambda^*, \mu_1^*, \ldots, \mu_K^*)$ that corresponds to the constrained optimum of the original problem, but we will use it in an alternative manner to reach our goal. Applying the stationarity conditions, any constrained solution $(v^*, \lambda^*, \mu_1^*, \ldots, \mu_K^*)$ will satisfy the following equations:

$$\begin{cases} \dfrac{\partial \mathcal{J}}{\partial \lambda}(v^*, \lambda^*, \mu_1^*, \ldots, \mu_K^*; \theta) = 0. \\ \dfrac{\partial \mathcal{J}}{\partial \mu_k}(v^*, \lambda^*, \mu_1^*, \ldots, \mu_K^*; \theta) = 0. \end{cases}$$

Let us focus on the first partial derivative. Note that it also corresponds to the derivative of a functional with respect to a trajectory, so it needs to be carefully handled. In fact, that partial derivative is an alternative representation of the Euler-Lagrange stationarity equations

[66]; the equations are the direct result of the Gateaux derivatives of $\mathcal{J}$ being 0 at v^* for any admissible direction function η. In other words,

$$\delta\mathcal{J}[v^*](\eta) = 0,$$

for every admissible trajectory η. Before continuing further on, it is important to determine what admissible means in this context. In optimal control theory, a variation or perturbation η is said to be admissible if for all sufficiently small $\epsilon > 0$, $v_\epsilon(t) = v^*(t) + \epsilon\eta(t)$ is still feasible, i.e., $\exists\epsilon_0 : v_\epsilon(t) \in A, \forall\epsilon > 0$ that satisfies $\epsilon < \epsilon_0$, where A is the feasibility set of the control problem [67]. It is specifically important to note that $\eta(0) = \eta(T) = 0$ and that the function must be continuous in intervals where $v(t)$ is not, i.e., $\eta(t_k^+) = \eta(t_k^-)$ for every spiking time step t_k.

Therefore, let us consider any admissible variation η and calculate the Gateaux derivative of $\mathcal{J}$ at any function v for η:

$$\begin{aligned}
\delta\mathcal{J}[v](\eta) &= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\big[\mathcal{J}(v+\epsilon\eta) - \mathcal{J}(v)\big] \\
&= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\big[\mathcal{L}(v+\epsilon\eta;\theta) - \mathcal{L}(v;\theta)\big] \\
&\quad + \lim_{\epsilon\to 0}\frac{1}{\epsilon}\left[\int_0^T \lambda(t)\left(\frac{d}{dt}\left[v(t)+\epsilon\eta(t)\right] - f(v(t)+\epsilon\eta(t);\theta) - \frac{dv}{dt}(t) + f(v(t);\theta)\right)dt\right] \\
&\quad + \lim_{\epsilon\to 0}\frac{1}{\epsilon}\left[\sum_{k=1}^K \mu_k\big(v(t_k^+) + \epsilon\eta(t_k^+) - v(t_k^+)\big)\right].
\end{aligned}$$

By developing each of the three limits, we have that:

- The third term can be easily solved:

$$\lim_{\epsilon\to 0}\frac{1}{\epsilon}\left[\sum_{k=1}^K \mu_k(v(t_k^+) + \epsilon\eta(t_k^+) - v(t_k^+))\right] = \sum_{k=1}^K \mu_k\eta(t_k^+).$$

- The first term is also relatively simple to derive, since it corresponds to the definition of the Gateaux derivative of $\mathcal{L}$ at v for the direction η. By applying the chain rule, we can employ the results we previously obtained to calculate that limit:

$$\begin{aligned}
\lim_{\epsilon\to 0}\frac{1}{\epsilon}[\mathcal{L}(v+\epsilon\eta;\theta) - \mathcal{L}(v;\theta)] = \delta\mathcal{L}[v](\eta) &= \frac{\partial\mathcal{L}}{\partial\hat{y}}(\hat{y})\cdot\frac{d\hat{y}}{du}(u)\cdot\delta u[v](\eta) \\
&= \int_0^T \kappa(\hat{y}-y)\sigma'(v(t))\eta(t)dt = \int_0^T \frac{\delta\mathcal{L}}{\delta v}(t)\eta(t)dt.
\end{aligned}$$

- Deriving the second term requires interchanging the limit and the integral (which can be done by the same argument introduced in the previous subsection) and integrating by parts the resulting function:

$$\lim_{\epsilon\to 0}\frac{1}{\epsilon}\left[\int_0^T \lambda(t)\left(\frac{d}{dt}\left[v(t)+\epsilon\eta(t)\right] - f(v(t)+\epsilon\eta(t);\theta) - \frac{dv}{dt}(t) + f(v(t);\theta)\right)dt\right].$$

We shall now interchange the derivative and the limit, simplify $\frac{\epsilon\eta(t)}{\epsilon}$ and factor out a minus sign by reversing the sign of each term:

$$\begin{aligned}
&\lim_{\epsilon\to 0}\frac{1}{\epsilon}\left[\int_0^T \lambda(t)\left(\frac{d}{dt}\left[v(t)+\epsilon\eta(t)\right]-f(v(t)+\epsilon\eta(t);\theta)-\frac{dv}{dt}(t)+f(v(t);\theta)\right)dt\right]\\
&=-\int_0^T \lambda(t)\left(\lim_{\epsilon\to 0}\left[\frac{f(v(t)+\epsilon\eta(t);\theta)-f(v(t);\theta)}{\epsilon}\right]-\frac{d\eta}{dt}(t)\right)dt\\
&=-\int_0^T \lambda(t)\left(\lim_{\epsilon\to 0}\left[\frac{f(v(t)+\epsilon\eta(t);\theta)-f(v(t);\theta)}{\epsilon}\right]\right)dt+\int_0^T \lambda(t)\frac{d\eta}{dt}(t)dt.
\end{aligned}$$

Taking $x=v(t)$ and $h=\epsilon\eta(t)$,

$$\begin{aligned}
\lim_{\epsilon\to 0}\left[\frac{f(v(t)+\epsilon\eta(t);\theta)-f(v(t);\theta)}{\epsilon}\right]&=\lim_{h\to 0}\left[\frac{f(x+h;\theta)-f(x;\theta)}{h}\eta(t)\right]\\
&=\frac{df}{dx}(x)\eta(t)=\frac{df}{dv}(v(t))\eta(t).
\end{aligned}$$

Therefore, the initial limit can be simplified to

$$-\int_0^T \lambda(t)\frac{df}{dv}(v(t))\eta(t)dt+\int_0^T \lambda(t)\frac{d\eta}{dt}(t)dt.$$

To conclude, we shall apply integration by parts to the second integral after decomposing it as a series of integrals that cover each subinterval (t_k, t_{k+1}). Applying integration by parts will require evaluating the integrand at the integration bounds, but note that the lower bound corresponds to t_k^+, whereas the upper bound corresponds to t_{k+1}^-. This is very important, as each integral covers the subinterval (t_k, t_{k+1}), and as such, the actual bounds are t_k^+ and t_{k+1}^-, respectively. Therefore,

$$\begin{aligned}
\int_0^T \lambda(t)\frac{d\eta}{dt}(t)dt&=\sum_{k=0}^{K-1}\int_{t_k}^{t_{k+1}} \lambda(t)\frac{d\eta}{dt}(t)dt\\
&=\lambda(t_1^-)\eta(t_1^-)+\sum_{k=1}^{K-1}\left[\lambda(t_{k+1}^-)\eta(t_{k+1}^-)-\lambda(t_k^+)\eta(t_k^+)\right]\\
&\quad-\lambda(t_k^+)\eta(t_k^+)-\int_0^T \frac{d\lambda}{dt}(t)\eta(t)dt\\
&=\sum_{k=1}^{K}[\lambda(t_k^-)-\lambda(t_k^+)]\eta(t_k^+)-\int_0^T \frac{d\lambda}{dt}(t)\eta(t)dt.
\end{aligned}$$

Therefore, the second term can be written as

$$\int_0^T\left[-\lambda(t)\frac{df}{dv}(v(t))-\frac{d\lambda}{dt}(t)\right]\eta(t)dt+\sum_{k=1}^{K}[\lambda(t_k^-)-\lambda(t_k^+)]\eta(t_k^+).$$

Now that we have computed each term, we can combine them all to provide the Gateaux derivative of $\mathcal{J}$ at v for any admissible trajectory η, given by Equation (4.13):

$$\delta\mathcal{J}[v](\eta) = \int_0^T \left[\frac{\delta\mathcal{L}}{\delta v}(t) - \lambda(t)\frac{df}{dv}(v(t)) - \frac{d\lambda}{dt}(t)\right]\eta(t)dt + \sum_{k=1}^{K}[\lambda(t_k^-) - \lambda(t_k^+) + \mu_k]\eta(t_k^+). \quad (4.13)$$

We will now apply the stationarity condition to prove two essential properties of the previous Gateaux derivative. Let

$$\begin{cases} F(t) = \dfrac{\delta\mathcal{L}}{\delta v^*}(t) - \lambda^*(t)\dfrac{df}{dv}(v^*(t)) - \dfrac{d\lambda^*}{dt}(t). \\ G_k = \lambda^*(t_k^-) - \lambda^*(t_k^+) + \mu_k^*. \end{cases}$$

Then, $F(t) = 0$ and $G_k = 0,\ \forall k \in 1, \ldots, K$. We shall prove both results by contradiction:

- Suppose $G_k \neq 0$. That implies that there exists some k_0 for which $G_{k_0} \neq 0$. Let us consider the following admissible needle variation:

  $$\eta_\epsilon(t) = \begin{cases} \dfrac{1}{\epsilon}, & \text{if } t \in (t_{k_0}, t_{k_0} + \epsilon) \\ 0, & \text{else} \end{cases}, \quad \text{such that } t_{k_0} + \epsilon < t_{k_0+1}$$

  Note that for that variation, $\sum_{k=1}^{K} G_k\eta_\epsilon(t_k^+) = \frac{G_{k_0}}{\epsilon}$ holds. Moreover,

  $$\int_0^T F(t)\eta_\epsilon(t)dt = \int_{t_{k_0}}^{t_{k_0}+\epsilon} F(t)\eta_\epsilon(t)dt \leq \sup_{0\leq t\leq T}|F(t)|\epsilon.$$

  These two properties ensure that

  $$\lim_{\epsilon\to 0}\delta\mathcal{J}[v^*](\eta_\epsilon) = \lim_{\epsilon\to 0}\left[\int_0^T F(t)\eta_\epsilon(t)dt + \sum_{k=1}^{K} G_k\eta_\epsilon(t_k^+)\right] = \lim_{\epsilon\to 0}\frac{G_{k_0}}{\epsilon} \neq 0.$$

  However, this leads to a contradiction, since any solution of the constrained problem must satisfy $\delta\mathcal{J}[v^*](\eta) = 0$ for any admissible variation η. Therefore, $G_k = 0$, $\forall k \in 1, \ldots, K$.

- Likewise, suppose $F(t) \neq 0$. This implies that for some t_0 that does not correspond to a spiking event, $F(t_0) \neq 0$. For simplicity, suppose $F(t_0) > 0$ (the other case is analogous). Note that $t_k < t_0 < t_{k+1}$ for some k, and therefore, F is continuous in an interval $[t_0 - \delta, t_0 + \delta] \subset (t_k, t_{k+1})$. This implies there exists some value $c > 0$ where

  $$F(t) \geq c > 0, \forall t \in [t_0 - \delta, t_0 + \delta].$$

  We shall now consider the following admissible variation:

  $$\eta(t) = \begin{cases} \left[1 - \left(\dfrac{t - t_0}{\delta}\right)\right]^2, & \text{if } |t - t_0| < \delta. \\ 0, & \text{else.} \end{cases}$$

Note that $\eta(t_k^+) = 0$ for every spiking event and that $\eta(t) \geq 0$ (especially $\eta(t) > 0$ for $t \in (t_0 - \delta, t_0 + \delta)$). Therefore,

$$\begin{aligned}\delta\mathcal{J}[v^*](\eta) &= \int_0^T F(t)\eta(t)dt + \sum_{k=1}^{K} G_k\eta(t_k^+) = \int_0^T F(t)\eta(t)dt \\ &= \int_{t_0-\delta}^{t_0+\delta} F(t)\eta(t)dt \geq c\int_{t_0-\delta}^{t_0+\delta}\eta(t)dt > 0.\end{aligned}$$

Again, this is a contradiction, and therefore, $F(t) = 0$.

However, why are the previous two results relevant? It turns out that they play a key role in computing the partial derivatives of the loss with respect to the learnable parameters efficiently. On the one hand, $F(t) = 0$ implies that

$$F(t) = \frac{\delta\mathcal{L}}{\delta v^*}(t) - \lambda^*(t)\frac{df}{dv}(v^*(t)) - \frac{d\lambda^*}{dt}(t) = 0 \Longrightarrow \boxed{\frac{d\lambda^*}{dt}(t) = -\lambda^*(t)\frac{df}{dv}(v^*(t)) + \frac{\delta\mathcal{L}}{\delta v^*}(t)}.$$

This is known as the differential equation of the adjoint state, and it is valid for $t \neq t_k$. The terminal condition $\lambda^*(T) = 0$ must also hold, because there is no terminal cost for $v(T)$; the loss is calculated by integrating the whole trajectory rather than considering the final value. On the other hand, $G_k = 0$ implies that

$$G_k = 0 \Longrightarrow \mu_k = \lambda(t_k^+) - \lambda(t_k^-).$$

As for $\lambda^*(t)$, solving the differential equation and considering the spiking time steps t_k and the terminal condition, we should obtain a function satisfying

$$\begin{cases}\lambda^*(t) &= \text{solveODE}(F(t) = 0). \\ \lambda^*(T) &= 0. \\ \lambda^*(t_k^-) = \frac{\partial\phi}{\partial v^{*-}}(v^*)\lambda^*(t_k^+) = 0.\end{cases}$$

$\lambda^*(t)$ is something we can efficiently compute in the backward pass by integrating its ODE (since $v^*(t)$ will be the solution we retrieve by solving the differential equation of the neural model, we might refer to λ^* as λ and v^* as v). The advantage of using the adjoint state is that the gradient of the loss with respect to the integration parameters corresponds to the expression in Equation (4.14):

$$\nabla_{w_1,\ldots,w_n,b}\,\mathcal{L}(w_1,\ldots,w_n,b) = -\int_0^T \left(\frac{\partial f}{\partial w_1}(t,w_1),\ldots,\frac{\partial f}{\partial w_n}(t,w_n),\frac{\partial f}{\partial b}(t,b)\right)\lambda(t)dt. \quad (4.14)$$

To show this, let us consider the original formula for the gradient of the loss with respect to the learnable parameters:

$$\nabla_{w_1,\ldots,w_n,b}\,\mathcal{L}(w_1,\ldots,w_n,b) = \int_0^T \frac{\delta\mathcal{L}}{\delta v}(t)\left(\frac{\partial v}{\partial w_1}(t,w_1),\ldots,\frac{\partial v}{\partial w_n}(t,w_n),\frac{\partial v}{\partial b}(t,b)\right).$$

Now, let us substitute $\frac{\delta \mathcal{L}}{\delta v}(t)$ for the terms in the adjoint differential equation:

$$\begin{aligned}
&\nabla_{w_1,\ldots,w_n,b}\, \mathcal{L}(w_1,\ldots,w_n,b) \\
&= \int_0^T \frac{\delta \mathcal{L}}{\delta v}(t) \left(\frac{\partial v}{\partial w_1}(t,w_1),\ldots,\frac{\partial v}{\partial w_n}(t,w_n), \frac{\partial v}{\partial b}(t,b) \right) dt \\
&= \int_0^T \left(\frac{d\lambda}{dt}(t) + \lambda(t)\frac{df}{dv}(v(t)) \right) \left(\frac{\partial v}{\partial w_1}(t,w_1),\ldots,\frac{\partial v}{\partial w_n}(t,w_n), \frac{\partial v}{\partial b}(t,b) \right) dt \\
&= \int_0^T \frac{d\lambda}{dt}(t) \left(\frac{\partial v}{\partial w_1}(t,w_1),\ldots,\frac{\partial v}{\partial w_n}(t,w_n), \frac{\partial v}{\partial b}(t,b) \right) dt \\
&\quad + \int_0^T \lambda(t)\frac{df}{dv}(v(t)) \left(\frac{\partial v}{\partial w_1}(w_1),\ldots,\frac{\partial v}{\partial w_n}(w_n), \frac{\partial v}{\partial b}(b) \right) dt.
\end{aligned}$$

Let us refer to each integral as I_1 and I_2, respectively. Now, let us consider the following components:

- We shall first integrate I_1 by parts by selecting

$$u(t) = \left(\frac{\partial v}{\partial w_1}(t,w_1),\ldots,\frac{\partial v}{\partial w_n}(t,w_n), \frac{\partial v}{\partial b}(t,b) \right);\ v(t) = \lambda(t).$$

  Therefore,

$$\begin{aligned}
I_1 &= \lambda(t) \left(\frac{\partial v}{\partial w_1}(t,w_1),\ldots,\frac{\partial v}{\partial w_n}(t,w_n), \frac{\partial v}{\partial b}(t,b) \right)\Bigg|_0^T \\
&\quad - \int_0^T \lambda(t) \left(\frac{d}{dt}\left[\frac{\partial v}{\partial w_1}\right](t,w_1),\ldots,\frac{d}{dt}\left[\frac{\partial v}{\partial w_n}\right](t,w_n), \frac{d}{dt}\left[\frac{\partial v}{\partial b}\right](t,b) \right) dt \\
&= \lambda(T) \left(\frac{\partial v}{\partial w_1}(T,w_1),\ldots,\frac{\partial v}{\partial w_n}(T,w_n), \frac{\partial v}{\partial b}(T,b) \right) \\
&\quad - \lambda(0) \left(\frac{\partial v}{\partial w_1}(0,w_1),\ldots,\frac{\partial v}{\partial w_n}(0,w_n), \frac{\partial v}{\partial b}(0,b) \right) \\
&\quad - \int_0^T \lambda(t) \left(\frac{d}{dt}\left[\frac{\partial v}{\partial w_1}\right](t,w_1),\ldots,\frac{d}{dt}\left[\frac{\partial v}{\partial w_n}\right](t,w_n), \frac{d}{dt}\left[\frac{\partial v}{\partial b}\right](t,b) \right) dt.
\end{aligned}$$

  Since by the boundary constraint $\lambda(T) = 0$ and voltage not being dependent on the learnable parameters at $t = 0$ (as it is set to the resting state no matter which the values of w_i and b are), the previous expression simplifies to

$$- \int_0^T \lambda(t) \left(\frac{d}{dt}\left[\frac{\partial v}{\partial w_1}\right](t,w_1),\ldots,\frac{d}{dt}\left[\frac{\partial v}{\partial w_n}\right](t,w_n), \frac{d}{dt}\left[\frac{\partial v}{\partial b}\right](t,b) \right) dt.$$

- Recall that $\frac{dp_{w_i}}{dt}(t) = \frac{\partial v}{\partial w_i}(t,w_i) = \frac{\partial f}{\partial v}(t)p_{w_i}(t) + \frac{\partial f}{\partial w_i}(t,w_i)$ and that $\frac{dp_b}{dt}(t) = \frac{\partial v}{\partial b}(t,b)$ $= \frac{\partial f}{\partial v}(t)p_b(t) + \frac{\partial f}{\partial b}(t,b)$. By substituting those expressions into I_1, we have that

$$I_1 = -\int_0^T \lambda(t) \left(\frac{\partial f}{\partial v}(v(t))\frac{\partial v}{\partial w_1}(t,w_1) + \frac{\partial f}{\partial w_1}(t,w_1),\ldots,\frac{\partial f}{\partial v}(v(t))\frac{\partial v}{\partial b}(t,b) + \frac{\partial f}{\partial b}(t,b) \right) dt.$$

Adding I_1 and I_2, we will obtain the gradient of the loss:

$$\begin{aligned}
I_1 + I_2 &= -\int_0^T \lambda(t) \left(\frac{\partial f}{\partial v}(v(t)) \frac{\partial v}{\partial w_1}(t) + \frac{\partial f}{\partial w_1}(t, w_1), \ldots, \frac{\partial f}{\partial v}(v(t)) \frac{\partial v}{\partial b}(t) + \frac{\partial f}{\partial b}(t, b) \right) dt \\
&\quad + \int_0^T \lambda(t) \frac{\partial f}{\partial v}(v(t)) \left(\frac{\partial v}{\partial w_1}(t, w_1), \ldots, \frac{\partial v}{\partial b}(t, b) \right) dt \\
&= \int_0^T \lambda(t) \left(-\frac{\partial f}{\partial v}(v(t)) \frac{\partial v}{\partial w_1}(t, w_1) + \frac{\partial f}{\partial v}(v(t)) \frac{\partial v}{\partial w_1}(t, w_1) - \frac{\partial f}{\partial w_1}(t, w_1), \ldots, \right. \\
&\qquad \left. -\frac{\partial f}{\partial v}(v(t)) \frac{\partial v}{\partial b}(t, b) + \frac{\partial f}{\partial v}(v(t)) \frac{\partial v}{\partial b}(t, b) - \frac{\partial f}{\partial b}(t, b) \right) dt \\
&= \int_0^T \lambda(t) \left(-\frac{\partial f}{\partial w_1}(t, w_1), \ldots, -\frac{\partial f}{\partial w_n}(t, w_n), -\frac{\partial f}{\partial b}(t, b) \right) dt \\
&= -\int_0^T \lambda(t) \left(\frac{\partial f}{\partial w_1}(t, w_1), \ldots, \frac{\partial f}{\partial w_n}(t, w_n), \frac{\partial f}{\partial b}(t, b) \right) dt.
\end{aligned}$$

That is the result we were looking for. But why is it that the expression is convenient when it comes to computing the partial derivatives of the loss with respect to the learnable parameters? Note that the partial derivatives of f with respect to the learnable parameters are trivial to calculate, since $\frac{\partial f}{\partial w_i}(t, w_i) = x_i(t)$ and $\frac{\partial f}{\partial b}(t, b) = 1$. Therefore, the gradient of the loss with respect to w_i and b can be simplified to the expression shown in Equation (4.15):

$$\nabla_{w_1,\ldots,w_n,b} \mathcal{L}(w_1, \ldots, w_n, b) = -\int_0^T \lambda(t)(x_1(t), \ldots, x_n(t), 1)dt. \tag{4.15}$$

As for the learnable parameter ϕ, we can also find a simple closed formula employing the adjoint state. Let us differentiate the Lagrangian function with respect to ϕ (considering any input variable):

$$\begin{aligned}
&\frac{\partial \mathcal{J}}{\partial \phi}(v, \lambda, \mu_1, \ldots, \mu_K; \theta) \\
&= \frac{\partial \mathcal{L}}{\partial \phi}(v; \theta) + \frac{\partial}{\partial \phi} \left[\int_0^T \lambda(t) \left(\frac{dv}{dt}(t) - f(v(t); \theta) \right) dt \right] + \frac{\partial}{\partial \phi} \left[\sum_{k=1}^K \mu_K(v(t_k^+) - \phi) \right] \\
&= \frac{\partial \mathcal{L}}{\partial \hat{y}}(\hat{y}) \cdot \frac{d\hat{y}}{du}(u) \cdot \frac{du}{dv}(v) \cdot \frac{dv}{d\phi}(\phi) - \sum_{k=1}^K \mu_k.
\end{aligned}$$

Since v is an independent variable, it does not depend on ϕ. Therefore, the first term will vanish, leaving

$$\frac{\partial \mathcal{J}}{\partial \phi}(v, \lambda, \mu_1, \ldots, \mu_K; \theta) = -\sum_{k=1}^K \mu_k.$$

However, after completing the forward pass, we will not have any trajectory $v(t)$, but rather one $v^*(t)$ that satisfies the constraints of the Lagrangian function. Moreover, such a function

will depend on ϕ. Therefore,

$$\frac{\partial \mathcal{J}}{\partial \phi}(v^*, \lambda^*, \mu_1^*, \dots, \mu_K^*; \theta) = \frac{\partial \mathcal{L}}{\partial \phi}(v^*; \theta).$$

For simplicity, since we will always obtain feasible trajectories, let us refer to $v^*(t)$ as $v(t)$. Considering the previous two results, the partial derivative of the loss with respect to ϕ comes down to

$$\frac{\partial \mathcal{L}}{\partial \phi}(w_1, \dots, w_n, \phi) = -\sum_{k=1}^{K} \mu_k.$$

But remember that we derived $\mu_k = \lambda(t_k^+) - \lambda(t_k^-)$ and that the adjoint reset condition states $\lambda(t_k^-) = 0$. Therefore, by combining everything, Equation (4.16) shows the gradient of the loss with respect to ϕ, which corresponds to its partial derivative:

$$\nabla_\phi \mathcal{L}(\phi) = \frac{\partial \mathcal{L}}{\partial \phi}(w_1, \dots, w_n, \phi) = -\sum_{k=1}^{K} (\lambda(t_k^+) - \lambda(t_k^-)) = -\sum_{k=1}^{K} \lambda(t_k^+). \tag{4.16}$$

Employing the adjoint results in a considerable gain in efficiency. First of all, we have closed and easily tractable formulas for the partial derivatives of the loss. They do not involve nested integrals or solving a set of ordinary differential equations that grows as the number of learnable parameters grows. We only need to solve the adjoint differential equation numerically, which translates to solving only a m-dimensional differential equation, m being the number of state variables; it does not scale with the number of learnable parameters. Therefore, this is a memory-efficient technique that only requires storing $\lambda(t)$, and any numerical integrator can be applied. It is a versatile approach.

Remark. The mathematical derivation of some of the components we have introduced has been built on top of the main results in [43, 59, 68, 69]. While they do not provide derivations as mathematically exhaustive as those in this chapter, we have developed our theoretical framework to obtain those results. Other components have been derived from scratch.

4.3 Comparative Analysis of the Biological Plausibility of the Model

To end with this chapter, we shall offer a critical analysis of the biological plausibility of the developed neural model, especially comparing it with current machine learning approaches. It will also serve as a summary of this chapter.

We have been working with a reformulation of the perceptron or the artificial neuron that includes an internal state (membrane voltage) that dictates how the neuron will respond to a given input. We have specified that membrane voltage will be characterised by the quadratic integrate-and-fire model, since being the topological normal form for the saddle-node bifurcation, it can represent a wide variety of phenomena found in integrator neurons.

However, as mentioned, we have inserted this module into the existing machine learning framework in order to make it compatible with existing training toolsets. Although computing the gradient of the loss with respect to the trainable parameters is mathematically exhaustive, the explicit formula obtained via the adjoint method shows that the training procedure can be implemented efficiently.

However, how biologically plausible is this formulation? Unlike typical machine learning neural models, where time is not inherently considered, this reformulation has time as a core component, enabling the neuron to dynamically respond to injected inputs. Moreover, the response follows a robust neurocomputational model, so we should expect a similar behaviour as the one shown in Figure 3.2. This incorporation allows us to bridge the gap between classic computational neuroscience and neural-based machine learning, showcasing that both fields are compatible with each other. We consider this to be an insightful conclusion, although we will first need to verify that the theoretical model we have developed can actually be implemented and that it exhibits coherent behaviour via some experiments. In any case, the theoretical results show great potential for developing artificial intelligence using spiking neural networks and neural ordinary differential equations.

However, every component of the approach is not realistic. Even this proposal lacks some considerable biological or neurocomputational aspects. To begin with, it does not inherently model resonator neurons. No matter how complex the ordinary differential equation is, we will not be able to model Andronov–Hopf bifurcations, leaving certain neuron types unaccounted for. In addition, employing backpropagation as the training scheme is not the best option with regard to biological plausibility. However, the main reason for relying on that technique is efficiency, as it offers an implementation-friendly and efficient framework for training our neuron. It also allows us to easily integrate this model into existing machine learning frameworks. In any case, exploring different training alternatives is a reasonable line of research.

This concludes the fourth chapter. So far, we have covered several theoretical concerns covering computational neuroscience, machine learning and this reformulation of the artificial neuron. It is now, therefore, mandatory to conduct some experiments to show whether theoretical results are consistent and to compare if this modelling we have developed actually has enough capacity to solve some basic machine learning tasks. Positive results may indicate that combining spiking neural networks with neural ordinary differential equations as in our proposal is a promising field on top of which to develop artificial intelligence.

CHAPTER 5

Implementation and Empirical Validation of the Model via Experimentation

In scientific research, theoretical models provide invaluable knowledge for understanding complex phenomena, but in some cases, they remain hypotheses until empirically validated. Conducting experiments allows us to test (and, if necessary, refine) the assumptions, parameters and predictions that underlie our theories. This thesis has covered so far the theory for a biologically reasonably plausible neural model that integrates both components from classic computational neuroscience and neural machine learning. However, we need to validate the theoretical results we have obtained and analyse the extent to which this reformulation of the artificial neuron exhibits greater capacity than current implementations of its classic counterpart. Moreover, we also need to analyse the efficiency of the training scheme, the effect of the hyperparameters (learning rate, κ gain, v_{peak}, etc.), whether training is stable, the need for regularisation, etc. While there are hundreds, if not thousands, of aspects one could address in this chapter, given our focus on the foundations of neural machine learning and deep learning throughout this thesis, we find it more informative to conduct a modest experimentation illustrating the key requirements a neural model should satisfy.

However, there is an important aspect that also needs to be addressed before conducting any experiment: the theoretical model needs to be implemented. While it may seem a reasonably easy task to do, as we already have the results that dictate how the model must be implemented, some aspects need to be considered. For instance, continuous signals can not be stored on a computer. That means that the trajectory to generate, $v(t)$, will have to be, instead, a discrete signal v_l. Moreover, data transformation procedures also have to be implemented.

Therefore, this chapter will cover both of those two fundamental aspects that we have

described. On the one hand, we will provide a complete implementation of the neural model, considering all the aspects needed to conduct training on top of the scheme defined in Chapter 4. On the other hand, we will propose a series of experiments to validate the implementation and verify whether the model performs as expected.

5.1 Implementation: Training Framework for the Neural Model

One major challenge this thesis has faced is related to the implementation of the developed theoretical model. All of the content that we have covered suggests that continuous functions are needed to support the signals that the model creates. However, such an implementation on computers can not exist, since they contain finite resources. As a consequence, implementing all the components covered in Chapter 4 has required meticulously adapting the typical training scheme for deep learning models to our requirements. Changes are not significant, in the sense that this model is compatible with the deep learning paradigm, but such adjustments have been essential in order for our model to be trainable, always trying to ensure numerical methods produce minimal error. A training framework that considers all the aforementioned details requires implementing an algorithm to **transform data into input signals**, determining how the **forward pass will be conducted** (implementing a robust numerical integration method for the ordinary differential equation and including the effect of the resets) and **implementing an effective and efficient backward pass through the adjoint method** (robust numerical integration of the adjoint differential equation is also relevant for this part).

5.1.1 Algorithms for the Transformation of Discrete Data into Signals

Transforming any set of instances $X = \{X_1, \ldots, X_m\}$ into a set of signals $X(t) = \{X_1(t), \ldots, X_m(t)\}$ requires including a temporal component in the data. In fact, since computational resources are finite, these signals will have to be discrete. As a consequence, our initial goal will require building a set $X_l = \{X_l^{(1)}, \ldots, X_l^{(m)}\}$, where $X_l = X(t_l)$, that will contain the temporal training data. Although there are different alternatives we could think of, in Chapter 4 we indicated that we would employ the zero-order hold modelling and the dynamical source of input technique as the transformation procedures. Both techniques allow converting discrete data into continuous or discrete signals, but, as the results will show, they offer different levels of integration capacity to the model; the neuron can achieve better or more complex representation structures depending on how input signals are created.

5.1.1.1 Discrete Zero-Order Hold Modelling

The zero-order hold modelling method is relatively intuitive, as it consists of building a rectangular signal with the feature vector of the instances. After defining a time range $[0, T]$, divided into L discrete and evenly spaced points, as well as a subset $[l_1, l_2] \subseteq \{0, \ldots, L-1\}$, given a feature vector $(x_1, \ldots, x_n)$, its rectangular signal will be defined as $(0, \ldots, 0),\ \forall l$

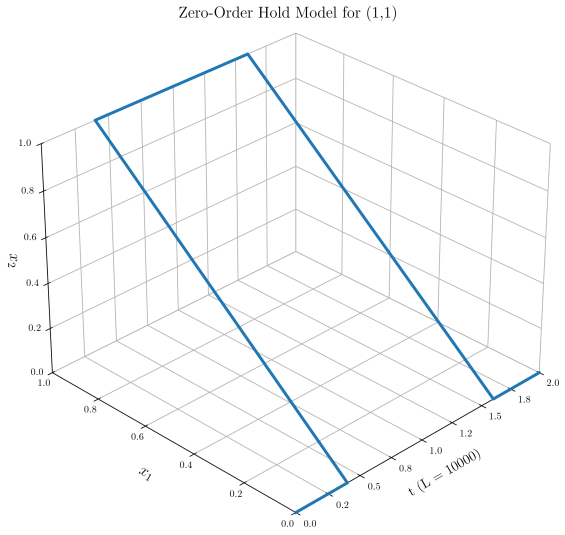

Figure 5.1: Zero-order hold model for $(x_1, x_2) = (1, 1)$. This example considers a time range $[0, 2]$ divided into $L = 10000$ discrete points, where $l_1 = 2000$ and $l_2 = 8000$. Other configurations are perfectly valid, but they might conceivably have an impact on training.

$\notin [l_1, l_2]$ and as $(x_1, \ldots, x_n),\ \forall l \in [l_1, l_2]$, or equivalently, as $(0, \ldots, 0),\ \forall t \notin [t_{l_1}, t_{l_2}]$ and as $(x_1, \ldots, x_n),\ \forall t \in [t_{l_1}, t_{l_2}]$. We could think of extending the feature vector along time, while also distinguishing intervals where the neuron will be stimulated and intervals where no input will be injected. Figure 5.1 shows a synthetic example of a zero-order hold model for an instance whose feature vector corresponds to $(1, 1)$. Note that other valid rectangular signals can be constructed by selecting other values for L, l_1 and l_2.

We think this is an intuitive approach to facilitate learning, as exposing the neuron to stimuli (defined by the feature vectors) over time will enable it to decide whether it should fire or not, or at least, whether it should fire more or less frequently. A successful training should enable the neuron to assign those firing patterns to instances of different classes. In fact, the manner in which we have formulated the neuron's training scheme should result in frequent spiking for instances of the class 1 and no spiking or few narrow spikes for instances of the class 0. This is because the neuron is trained to maximise the integral of the signal $s(t)$ for instances of the class 1 and minimise the integral for instances of the class 0.

As a summary of this subsection, Algorithm 1 contains the pseudocode for the implementation of the transformation based on the zero-order hold technique.

5.1.1.2 Dynamical Source of Input

The second alternative we propose is an enhancement to the zero-order hold, where, instead of inputting a constant input in an interval $[l_1, l_2]$, that interval is split into contiguous windows of the same length and a minor transformation is applied (the same one for all the constant samples that lie in that window) to modulate the data, so that the neuron can understand and

Algorithm 1: Transformation of discrete feature vectors into the zero-order hold discrete model.

Input: $\{X_1, \ldots, X_m\}$, L, l_1, l_2. $0 \leq l_1 \leq l_2 \leq L-1$ must hold.
Output: $\{X_l^{(1)}, \ldots, X_l^{(m)}\}$.
for $X_i \in \{X_1, \ldots, X_m\}$ **do**
  $X_l^{(i)} \leftarrow [0, \ldots, 0] \times L$;
  for $j \leftarrow l_1$ **to** l_2 **do**
    $X_j^{(i)} = [x_1^{(i)}, \ldots, x_n^{(i)}]$;
  end
end
return $\{X_l^{(1)}, \ldots, X_l^{(m)}\}$;

incorporate the time-dependent input into its rich dynamics. Therefore, for each feature of an instance, instead of having a single interval $[l_1, l_2]$ in which a constant value is injected into the neuron, we will have S intervals, each carrying a different constant value. Moreover, the signal will evolve gradually from one window to the next. Figure 5.2 shows an example of the dynamical source of input transformation for a synthetic instance whose feature vector corresponds to $(1, 1)$. The example considers $L = 10000$, $S = 12$, $f_1^{(i)}(x_1) = \cos\left(\frac{2\pi i}{S}\right) x_1$ and $f_2^{(i)}(x_2) = \sin\left(\frac{2\pi i}{S}\right) x_2$, $\forall i \in \{0, \ldots, S-1\}$. Dot-sinusoidal transformations enable smooth transitions between adjacent windows, so we believe they can be suitable candidates.

As a summary of this subsection, Algorithm 2 contains the pseudocode for the implementation of the transformation based on the dynamical source of input technique.

Algorithm 2: Transformation of discrete feature vectors via the dynamical source of input technique.

Input: $\{X_1, \ldots, X_m\}$, L, l_1, l_2, S, $\left\{f_1^{(1)}, \ldots, f_n^{(S)}\right\}$. $0 \leq l_1 \leq l_2 \leq L-1$ must hold.
Output: $\{X_l^{(1)}, \ldots, X_l^{(m)}\}$.
$interv\_len \leftarrow (l_2 - l_1 + 1)//S$;
for $X_i \in \{X_1, \ldots, X_m\}$ **do**
  $X_l^{(i)} \leftarrow [0, \ldots, 0] \times L$;
  for $j \leftarrow l_1$ **to** l_2 **do**
    for $s \leftarrow 0$ **to** $S-1$ **do**
      $inf\_limit \leftarrow j + s \cdot interv\_len$;
      $X^{(i)}_{inf\_lim:inf\_lim+interv\_len} = [f_1^{(s)}(x_1^{(i)}), \ldots, f_n^{(s)} x_n^{(i)}]$;
    end
  end
end
return $\{X_l^{(1)}, \ldots, X_l^{(m)}\}$;

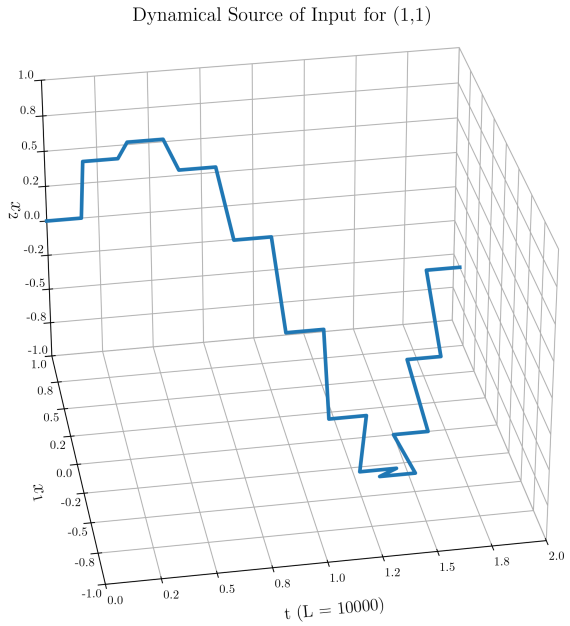

Figure 5.2: Dynamical source of input for $(x_1, x_2) = (1, 1)$. This example considers a time range $[0, 2]$ divided into $L = 10000$ discrete points, where $l_1 = 0$ and $l_2 = 9999$. In addition, $S = 12$, $f_1^{(i)}(x_1) = \cos\left(\frac{2\pi i}{S}\right) x_1$ and $f_2^{(i)}(x_2) = \sin\left(\frac{2\pi i}{S}\right) x_2$, $\forall i \in \{0, \ldots, S-1\}$.

5.1.2 Algorithm for the Forward Pass

The forward pass of the model consists of computing a value $\hat{y}$ from an input $X_l^{(i)}$. However, this requires solving some intermediate steps that we need to contemplate. First of all, a differential equation has to be integrated to obtain the signal $v(t)$, which corresponds to the evolution of the membrane voltage over time. That signal has then to be transformed into the shifted instantaneous firing rate signal $s(t)$ by applying a sigmoid and subtracting 0.5. To end with, we need to integrate $s(t)$ and apply a final sigmoid to obtain the class prediction confidence. However, these are not the only details that we need to consider, since the backward pass requires employing both $v(t)$ and the set of spike time steps $\{t_1, \ldots, t_K\}$ to optimise the learnable parameters. Therefore, the forward pass must also register when spikes occur so as to output $\hat{y}$, $v(t)$ and $\{t_1, \ldots, t_K\}$.

Now that we have detailed all the aspects to consider in order to implement the forward pass, it may seem reasonably easy to devise an algorithm that does everything we have mentioned. However, since we can only treat signals as discrete objects, there are now new aspects that we also need to consider.

First, the ordinary differential equation corresponding to the quadratic integrate-and-fire model has to be numerically integrated to obtain $v(t)$. One of the simplest and most implementation-friendly numerical integration techniques is the Euler method [70], developed by Leonhard Euler. This is a first-order numerical procedure, and while there are various other

techniques that achieve better error resolution, such as the Runge–Kutta–Fehlberg method [24], their implementation is not that simple. Therefore, provided that we select a sufficiently small $\Delta t = \frac{T}{L-1}$ for the discretisation time step size, we will obtain reasonably precise approximations while keeping the implementation simple. The forward pass will be adjusted to this method, but since this method can be employed to numerically integrate n-dimensional differential equations, we find it to be a simple yet effective method. However, analysing the precision this method offers should be important as a future line of research. As for how the Euler method works, let $\frac{dv}{dt}(t) = f(t, v(t))$ be the differential equation to numerically integrate. The objective will be to find a discrete function v_l such that $v_l \approx v(t_l)$, $\forall l \in \{0, \dots, L-1\}$. We will consider evenly spaced samples in principle, i.e., $t_l - t_{l-1} = t_{l+1} - t_l$, $\forall l \in [1, L-2]$, and therefore use the same Δt, but different discretisation steps can also be included, provided that v_l is consistently defined. In such a scenario, v_l will be computed as Equation (5.1) shows:

$$\begin{cases} v_0 &= v(0). \\ v_{l+1} &= v_l + \Delta t \; f(t_l, v_l). \end{cases} \tag{5.1}$$

Since in our case $v(0) = \phi$, we have a valid method to approximate $v(t)$ through v_l.

Nevertheless, there is an additional issue that we need to address. The quadratic integrate-and-fire model determines that voltage must be set to ϕ once $v(t)$ reaches v_{peak}. If we only integrate the ordinary differential equation via the Euler method without considering the reset constraint, we will not obtain any spiking behaviour at all. In fact, it will be the case that, when voltage reaches its peak value, we will find that for some value of l, $v_l < v_{\text{peak}}$ but $v_{l+1} > v_{\text{peak}}$. This means that somewhere between l and $l+1$, we will need to reset voltage and then integrate the differential equation from that point to $l+1$ in order to obtain the correct approximation for $v(t_{l+1})$. In order to solve this problem, we propose verifying at each iteration whether $v \geq v_{\text{peak}}$. If that condition is not satisfied, the differential equation will be integrated without additional considerations. However, if the condition is met, we will estimate the time step t_k in which the reset should have occurred via linear interpolation. Interpolation will be applied employing t_l, t_{l+1}, v_l and v_{l+1}, and t_k will be estimated as Equation (5.2) shows:

$$t_k \approx t_l + \frac{v_{\text{peak}} - v_l}{v_{l+1} - v_l}(t_{l+1} - t_l). \tag{5.2}$$

Note that interpolation introduces a new source of error, as well as that it does not consider multiple spiking events in the same time step, but taking a large value for L should minimise the impact of these two issues. After estimating t_k, v_l will be calculated employing Equation (5.1), but adjusting the bounds. This is, v_l will be calculated as follows:

$$v_{l+1} = \phi + (t_{l+1} - t_k) \; f(t_k, \phi).$$

Once v_{l+1} has been correctly estimated, this procedure will be repeated iteratively until v_l has been completely computed. At this point, we will have successfully retrieved v_l and $\{t_1, \dots, t_K\}$. To end with, in order to calculate $\hat{y}$, we will need to integrate $\sigma(v_l) - 0.5$ over $[0, T]$ and then multiply the previous result by a gain factor κ and apply a final sigmoid. We

Algorithm 3: Forward pass for our neural model for a single instance.

Input: $model$, $X_l^{(i)}$, L, $t_{\text{range}} = [0\Delta t, \ldots, (L-1)\Delta t]$, κ.
Output: $\hat{y}_i, v_l^{(i)}, \{t_1, \ldots, t_K\}$.

$W \leftarrow model.W$;
$b \leftarrow model.b$;
$\phi \leftarrow model.\phi$;
$v_l^{(i)} \leftarrow [0] \times L$;
$v_0^{(i)} \leftarrow \phi$;
$t_{\text{spikes}} \leftarrow []$;
for $l \leftarrow 0$ **to** $L-1$ **do**
    $t_l \leftarrow t_{\text{range}}[l]$;
    $t_{l+1} \leftarrow t_{\text{range}}[l+1]$;
    $\Delta t \leftarrow t_{l+1} - t_l$;
    $I_{\text{inj}} \leftarrow W X_l^{(i)}$;
    $dv \leftarrow b + \left(v_l^{(i)}\right)^2 + I_{\text{inj}}$;
    $v_{l+1} \leftarrow v_l + dv\Delta t$;
    if $v_{l+1} \geq v_{peak}$ **then**
        $t_k \leftarrow interpolate(t_l, t_{l+1}, v_l, v_{l+1}, v_{\text{peak}})$;
        $t_{\text{spikes}}.append(t_k)$;
        $\Delta t \leftarrow t_{l+1} - t_k$;
        $dv \leftarrow b + \phi^2 + I_{\text{inj}}$;
        $v_{l+1} \leftarrow \phi + dv\Delta t$;
    end
end
$u_i \leftarrow trapezoidal\_rule\left(\sigma\left(v_l^{(i)}\right) - 0.5, t_{\text{range}}\right)$;
$\hat{y}_i \leftarrow \sigma(\kappa u)$;
return $\hat{y}_i, v_l^{(i)}, t_{spikes}$;

propose applying the trapezoidal rule as the numerical integration method, since it provides a good balance between precision and efficiency. Equation (5.3) shows the formula for the approximation of the integral of $v(t)$:

$$\int_0^T s(t)dt = \int_0^T [\sigma(v(t)) - 0.5]dt \approx \sum_{l=1}^{L-1} \frac{s(t_{l-1}) + s(t_l)}{2}(t_l - t_{l-1}). \tag{5.3}$$

Every other characteristic of the forward pass is analogous to the forward pass of a classic deep learning model. For instance, the injected input that will be incorporated into the differential equation will be calculated via a linear combination of the parameters of the model and the input data at the discrete time step l. In any case, this concludes the implementation of the forward pass. As a summary of all the aspects that we have introduced, Algorithm 3 contains the pseudocode for the complete forward pass implementation.

5.1.3 Algorithm for the Backward Pass and Optimisation of Parameters

The last principal component that we need to implement is the backward pass and the optimisation of the learnable parameters via gradient descent. Unlike traditional deep learning, where the gradient of the loss can be computed by employing standard vector/matrix operations, this case requires solving the adjoint differential equation to obtain the gradient of the loss. In principle, now that we have introduced how numerical methods can be applied to integrate differential equations, as well as that interpolation techniques can be employed to insert reset constraints, it should not be especially tedious to understand how the backward pass can be implemented. Moreover, since we have access to the spike time steps $\{t_1, \dots, t_K\}$, we do not need to do additional calculations. Therefore, solving the adjoint differential equation is the main challenge in the backward pass.

At this point, the reader may have noticed that there is an issue that prevents applying the Euler method the way we have introduced it. Note that we know both the terminal condition of the adjoint variable, $\lambda(T) = 0$, and the reset constraints, $\lambda(t_k^-) = 0$, but in order to apply the Euler method, we need to know the value of $\lambda(0)$. How can we solve this problem? Fortunately, the Euler method can be applied to approximate $\lambda(t)$ via λ_l, as usual, but by taking $\lambda_{L-1} = \lambda(T)$ and integrating the differential equation backward [71]. In fact, if $\lambda(t)$ satisfies the differential equation

$$\frac{d\lambda}{dt}(t) = -\lambda(t)\frac{df}{dv}(v(t)) + \frac{\delta\mathcal{L}}{\delta v}(t),$$

then, its reflected function $\lambda'(t)$ will satisfy the following differential equation:

$$\frac{d\lambda'}{dt}(t) = \lambda'(t)\frac{df}{dv}(v(t)) - \frac{\delta\mathcal{L}}{\delta v}(t).$$

Remark. By reflected function we mean that $\lambda(t) = \lambda'(T - t)$. Therefore, $\lambda(0) = = \lambda'(T), \dots, \lambda(T) = \lambda'(0)$.

We can make use of this property to numerically integrate the previous differential equation by employing the Euler method after setting $\lambda'_0 = \lambda(T) = 0$; the integration method is applied exactly the same way. Therefore, for every value λ'_l we calculate, we will also have obtained the value of λ_{L-l}.

However, there are some additional aspects that we need to polish in order to effectively implement the backward pass. First of all, even though we know when resets must occur, we need to compute and store the values $\lambda(t_k^+)$, because the partial derivative of the loss with respect to ϕ requires computing those values. However, it is not that trivial to obtain the values $\lambda(t_k^+)$. If t_k lands between two values t_{l-1} and t_l, we will need to apply interpolation techniques to approximate $\lambda(t_k^+)$. Moreover, once $\lambda(t_k^+)$ has been computed, we will need to reset the adjoint variable by setting $\lambda(t_k^-) = 0$ and continue to integrate from t_k^- to t_{l-1} (recall that we are integrating the differential equation backward) in order to obtain λ_{l-1}. In

Algorithm 4: Backward pass for our neural model for a single instance and optimisation of learnable parameters via gradient descent.

Input: $model$, y_i, $\hat{y}_i$, $X_l^{(i)}$, $\{t_1, \ldots, t_K\}$, L, t_{range}, $v_l^{(i)}$ κ, α.
Output: None.
$t_{\text{spikes}} \leftarrow reverse(t_{\text{spikes}})$;
$j\_sp \leftarrow 1$;
$lambda\_pluses \leftarrow []$;
$df\_dv \leftarrow 2v_l^{(i)}$;
$r \leftarrow \kappa(\hat{y}_i - y_i)\sigma'\left(v_l^{(i)}\right)$;
$\lambda_l \leftarrow [0] \times L$;
for $l \leftarrow L-1$ **to** 0 **do**
  $\Delta t \leftarrow t_l - t_{l-1}$;
  if $j\_sp \leq K$ **and** $(t_{l-1} \leq t_{spikes}[j\_sp] \leq t_l)$ **then**
    $t_k^+ \leftarrow t_{\text{spikes}}[j\_sp]$;
    $\lambda(t_k^+) \leftarrow \lambda_l + (t_l - t_k^+)(\lambda_l\, df\_dv[l] - r[l])$;
    $lambda\_pluses.append(\lambda(t_k^+))$;
    $df\_dv\_sp \leftarrow interpolate\left(t_{l-1}, t_l, df\_dv[l-1], df\_dv[l], \frac{df}{dv}(v(t_k^+)\right)$;
    $r\_sp \leftarrow interpolate\left(t_{l-1}, t_l, r[l-1], r_l, \frac{\delta\mathcal{L}}{\delta v}(t_k^+)\right)$;
    $\lambda_{l-1} \leftarrow 0 + (t_k^+ - t_{l-1})(0 \cdot df\_dv\_sp - r\_sp)$;
    $j\_sp \leftarrow j\_sp + 1$;
  else
    $\lambda_{l-1} \leftarrow \lambda_l + \Delta t(\lambda_l\, df\_dv[l] - r[l])$;
  end
  $integrand \leftarrow \lambda_l \cdot stack(X_l^{(i)}, 1)$;
  $\nabla_{w_1,\ldots,w_n,b}\mathcal{L} \leftarrow -trapezoid\_rule(integrand, t_{\text{range}})$;
  $\nabla_\phi \mathcal{L} \leftarrow -\sum_{k=1}^{K} lambda\_pluses[k]$;
  $model.W \leftarrow model.W - \alpha\nabla_{w_1,\ldots,w_n,b}\mathcal{L}[0:-1]$;
  $model.W \leftarrow model.W - \alpha\nabla_{w_1,\ldots,w_n,b}\mathcal{L}[-1]$;
  $model.\phi \leftarrow model.\phi - \alpha\nabla_\phi\mathcal{L}$;
end

case no spike lies in the interval $[t_{l-1}, t_l]$, we will apply the Euler method as usual:

$$\lambda_{l-1} = \lambda_l + (t_l - t_{l-1})\left(\lambda_l \frac{df}{dv}(v(t_l)) - \frac{\delta\mathcal{L}}{\delta v}(t_l)\right).$$

Note that in our case $\frac{df}{dv}(t) = 2v(t)$ and $\frac{\delta\mathcal{L}}{\delta v}(t) = \kappa(\hat{y} - y)\sigma'(v(t))$.

Once the adjoint has been computed, calculating the partial derivatives is straightforward; we only need to apply the formulas introduced in Chapter 4. In addition, optimising the learnable parameters requires applying a step of the gradient descent technique by employing

the partial derivatives that we have just calculated. As such, Algorithm 4 contains the pseudocode for the complete implementation of the backward pass for our model. It also includes the application of an iteration of gradient descent, where parameters are updated following the opposite direction of the gradient of the loss with respect to the parameters, as Equation (5.4) shows:

$$\theta_{j+1} = \theta_j - \alpha \nabla_{\theta_j} \mathcal{L}. \tag{5.4}$$

5.2 Experimentation I: Objectives and Suggested Methodology

Once the complete training scheme has been defined, we shall conduct different experiments to validate the proposal for our neural model. There are different aspects that we need to evaluate, not only how the model performs on some machine learning problems compared to typical implementations of the modern perceptron. For instance, we need to verify whether the model's internal state evolves following a spiking pattern typical in a quadratic integrate-and-fire formulation and whether the spiking patterns are coherent with the classes to predict.

However, resolution of machine learning tasks is also important. Linear classifiers like logistic regressors (essentially a perceptron for binary classification problems) can only separate data that are linearly separable. As a result, they hit a theoretical limit on nonlinear tasks, such as the XOR problem, where logistic regressors can do no better than 75 % accuracy on it. This problem consists of a dataset of four instances, $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$, where their corresponding labels are $0, 1, 1$ and 0, respectively, and a model has to classify those instances according to their labels. The concentric circumferences problem is another classic example where logistic regressors struggle to find a solution. In this case, we have a dataset of instances that correspond to the (x, y) coordinates of two concentric and different radius circumferences, and the goal is to assign each instance to its corresponding circumference.

At this point, the reader may have realised that if our reformulation of the perceptron were capable of behaving as expected and solving such problems, it would have significant implications for the conceptualisation of deep learning; a single neuron solving problems once deemed unsolvable would validate the hypothesis on which this thesis is built: that biologically inspired neurons can indeed enhance deep learning architectures. Naturally, we can not bring our proposal and the classic perceptron onto the same level, since our model performs more computation than calculating a linear combination, but at heart they are two models of the neuron tackling the same problems. Therefore, implications would be, at least, relevant.

As such, apart from testing the intended neural behaviour, we propose working on both the XOR problem and the concentric circumferences problem to test whether our model can handle linearly nonseparable problems. This will not guarantee that training will be simple or that any problem will be solvable with our neuron, but we will at least have robust empirical results that will support the capacity of this neuron to solve machine learning tasks with greater effectiveness than classic artificial neurons have. Therefore, we have devised a methodology that covers all these aspects, specified as follows to cover the objectives of the experimental section:

- The principal **goal consists of solving the XOR and the concentric circumferences problems**. It must be noted that we have chosen $r_0 = 1$ and $r_1 = 0.4$ as the hyperparameters for the problem, so any reference to a solution to this problem should be understood within that parameterisation and not as a general and universal solution to the concentric circumferences problem. Therefore, the first objective to address is the **evaluation of the performance of the model to solve linearly nonseparable binary classification problems**. As for the methodological aspects, we shall conduct 100 training trials for each problem, initialising the model with random configurations for the learnable parameters and evaluating whether the model approaches a local optimum that happens to be a valid solution for the problem. A solution will be valid if the set of parameters that has been obtained enables the model to achieve an accuracy of 100 % on the problem to solve. Therefore, **accuracy** has been selected as the effectiveness indicator. In addition, **decision boundaries** will be considered when explaining what the neuron has learnt.

- Some of the **aspects that may contribute to a more stable, unstable, effective, ineffective, etc. training will be compared** against common implementations of logistic regressors in *scikit-learn* [72], *Keras* [73] and *PyTorch* [74]. These include the hyperparameters introduced in the formulation of our neural model (gain factor and peak membrane voltage), as well as the number of training iterations and the value of the scoring metric. As for the methodological aspects, the effect of the hyperparameters on the performance of our neural model will be tested by considering different training trials (150 to 250 training epochs) on the XOR problem for the same set of parameters, as well as by running other trials where hyperparameters are adjusted one by one, leaving the other ones constant. On the other hand, comparison with typical implementations of logistic regressors will be conducted by launching 100 training trials where each model will be trained for 50 epochs, after which mean accuracy, maximum accuracy, training time per epoch and accuracy to training time ratio will be registered.

- **Verification of the intended neuron behaviour** is the last objective we seek to address. This will include analysing the spiking behaviour of the model, i.e., the shape of the function $v(t)$ when the neuron is asked to output a value corresponding to the class 0 versus when asked to output a value corresponding to the class 1. As for methodological aspects, we shall conduct training on a single synthetic instance for 50 epochs both when its assigned class is 0 and when it is 1, after which we will analyse whether membrane voltage evolution over time contains meaningful patterns.

5.3 Experimentation II: Analysis of the Obtained Results

The second part of this experimentation section will introduce the results we have obtained to answer the experimental concerns and objectives we have determined in Section 5.2.

Before delving into the results, we find it relevant to mention that this scheme allows for batched learning, where gradients are averaged for a set of instances before performing an

iteration of the gradient descent technique. Batched learning allows stabilising learning by reducing variance when the model has to be trained on a full dataset. However, since the datasets we have used are considerably small in size, batching will not be an important focus in this thesis. In addition, we have created the dataset corresponding to the concentric circumferences problem by selecting fifty evenly spaced points corresponding to one circumference and another fifty evenly spaced points corresponding to the second circumference. Finally, all experiments have been conducted by selecting $T = 2$, $L = 10000$, $l_1 = 0$ and $l_2 = L - 1$ (the rectangular signal is a straight line). Therefore, unless otherwise stated, those will be the values for those hyperparameters. This constitutes a line of research for adjusting those model-dependent hyperparameters, and it could be the case that, depending on the task, some values might be more suitable than others. However, this thesis will not cover such a broad analysis of hyperparameters. Instead, we have focused on more typical hyperparameters, such as the learning rate.

5.3.1 Verification of the Intended Neuron Behaviour

Let us begin by introducing the general aspects we have observed with regards to our neuron. One major concern we have had throughout the characterisation of this model has been whether the neuron would truly behave as a quadratic integrate-and-fire-like neuron while learning to produce distinct spiking patterns for each class once trained. In theory, if all the theoretical development were correct, training should result in loss descending and the neuron producing spiking patterns corresponding to each class; instances belonging to the class 0 should result in minor spiking activity, whereas instances belonging to the class 1 should produce continuous spiking activity in the neuron, indicating a firing response to such stimuli. Accordingly, and as commented in Section 5.2, in the first experiment we have conducted, we have trained the neuron on a single synthetic instance, assigning it to each class in different trials, and then we have evaluated whether its spiking activity matched our hypotheses. The experiment itself is simple, but obtaining positive results was crucial to validate the biological modelling capacity of our neuron from the very beginning. Since this is in principle a problem-agnostic property to verify, we have taken the instance $(x_1, x_2) = (1, 1)$ from the XOR problem, applied both transformation strategies and trained the neuron for 50 epochs both to predict class 0 and class 1. Other hyperparameters include $\kappa = 10$, $\alpha = 0.1$, $v_{\text{peak}} = 1$, $T = 2$, $L = 10000$, $l_1 = 0$, $l_2 = L - 1$ and $S = 12$ (when applicable) and the same dot-sinusoidal transformations employed to create Figure 5.2 (when applicable).

Figure 5.3 shows the results we have obtained for that experiment. The figure shows four different cases in which the model has received an input signal and where the evolution of membrane voltage has been registered. The top figures cover the two cases for the zero-order hold modelling technique, whereas the figures on the bottom correspond to the dynamical source of input technique. As can be seen, the results adjust to our expectations. Upon training the neuron to maximise the class confidence of an instance, it automatically deduces that for instances corresponding to the class 0, it should stay in a hyperpolarisation state and not produce a spiking signal, whereas for instances corresponding to the class 1, it deduces that it should perform spiking events to show that such instances produce positive stimuli in the

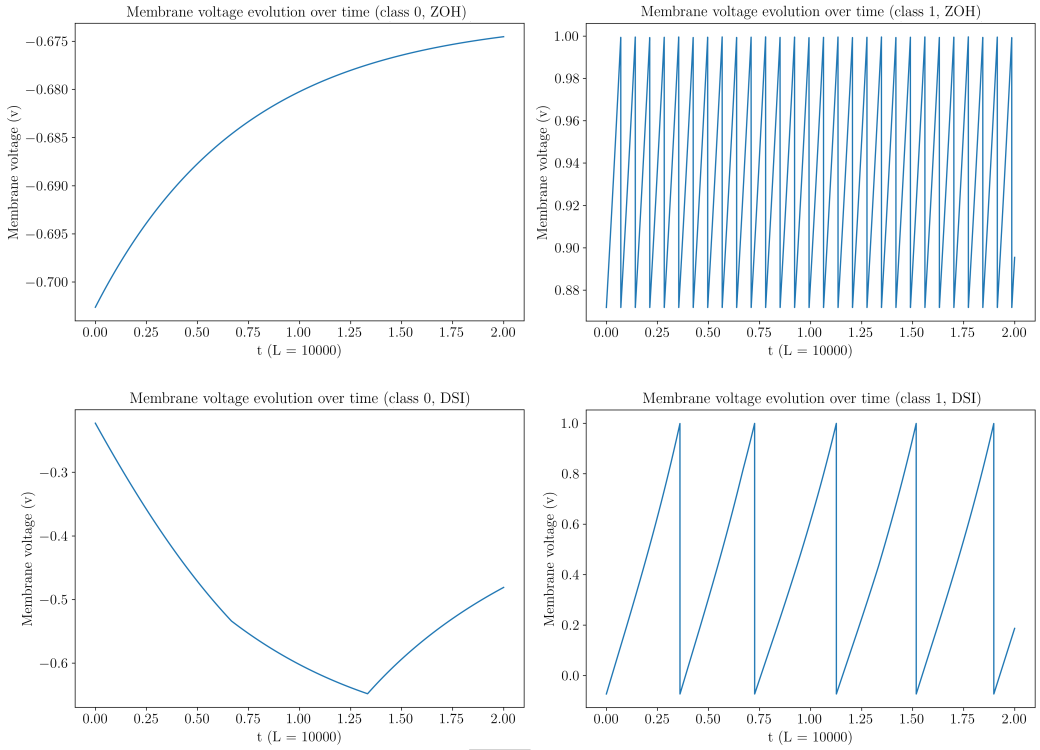

Figure 5.3: Behaviour of the neuron when trained to predict different classes. Top figures show the results when the zero-order hold model is applied, whereas bottom figures show the corresponding results when the dynamical source of input technique is applied.

neuron. We have a neural modelling formulation that behaves biologically plausibly and that exhibits the behaviour we designed it for! However, this is only the tip of the iceberg, and we need to see how the neuron handles datasets with multiple instances. Let us now address the particular results for each of the problems we have tackled.

5.3.2 Resolution of the Specified Problems

In this subsection, we will provide the details regarding the resolution of the XOR and the concentric circumferences problems.

5.3.2.1 Solution to the XOR Problem

After commenting on the general learning patterns of the neuron, we shall now focus on the resolution of the XOR problem. Before commenting on hyperparameters, training dynamics, etc., the most important question that needs to be addressed is whether our model is capable of solving the XOR problem or whether it exhibits suboptimal performance as logistic regressors do. In other words, does there exist a set $\{w_1^*, w_2^*, b^*, \phi^*\}$ for which the model achieves an accuracy of 1 in the XOR problem? After launching a set of 100 experiments or training

trials, where different models have been instantiated such that the parameters w_1 and w_2 were initialised at random following the Xavier initialisation procedure [75], whereas b and ϕ were initialised at random following a uniform distribution $\mathcal{U}(0,1)$, we have indeed found among those trials a set of parameters for which the neural model (setting $\kappa = 10$ and $v_{\text{peak}} = 1$, $\alpha = 0.001$ and batch size of 1) solves the XOR problem with an accuracy of 1: $w_1 = -0.51139857,\ w_2 = -0.36765103,\ b = 1.08252689$ and $\phi = -0.70497495$. This parameterisation corresponds to the zero-order hold modelling of the data, since we have observed that it yields a more robust convergence and simplifies training (at least in that it is more stable), even though the training procedure itself is not that simple. The dynamical source of input strategy offers a greater capacity to mix the dynamical components of the data, but training on top of it becomes more challenging and unstable, especially when increasing the value of S. In any case, a solution corresponding to the dynamical source of input is the following one: $w_1 = 0.5852366$, $w_2 = 0.76156609$, $b = 0.23454522$ and $\phi = -0.28512621$. Other hyperparameters include $\kappa = 10$, $v_{\text{peak}} = 0.6$, $\alpha = 0.001$, $S = 3$ and the aforementioned dot-sinusoidal transformations.

All in all, this result is significant because it shows that our biologically inspired neural model can optimally solve tasks that a classic artificial neuron can not. Although multilayer perceptrons handle XOR effortlessly, demonstrating that a properly structured single neuron can do the same reinforces the core hypotheses of this thesis. Figure 5.4 depicts the decision boundaries under zero-order hold and dynamical source of input modelling. Both approaches learn geometric separators that effectively partition the space, but the zero-order hold produces a strip-like boundary, valid for XOR yet consistently recurring in the concentric circumferences problem (where it is no longer valid as a solution), suggesting an input-method limitation on capacity. The dynamical source of input method also succeeds, albeit with a noisier boundary. In all cases, hyperparameter choices critically influence the clarity and quality of the learnt decision regions.

5.3.2.2 Solution to the Concentric Circumferences Problem

Once the XOR problem has been solved, we have tried to tackle a harder nonlinear artificial problem in order to show that our model can indeed solve challenging problems without the need to employ additional neurons or feature expansion (only the toolset provided by the neuron is allowed). The second problem we have tried to solve is the concentric circumferences problem, where two sets of instances are scattered around two concentric circumferences. Instances of one class will lie within the outer circumference, whereas instances of the other class will lie within the inner circumference. The objective is to find a decision boundary that effectively separates both structures. As commented earlier, we have constructed a dataset consisting of fifty points from one class and fifty from the other class, taking $r_0 = 1$ and $r_1 = 0.4$.

Does our model solve that concentric circumferences problem? Both possibilities would show different relevant aspects concerning the capacity of the model to address nonlinear problems. In this case, we have also found a valid solution for the problem, but only with the dynamical source of input strategy to model data. The solution includes $w_1 = -5.7601033$,

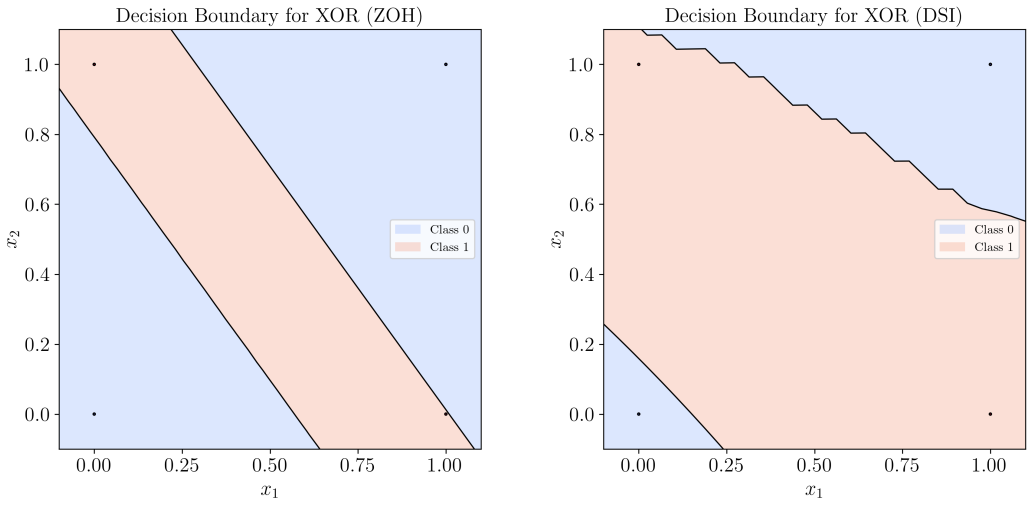

Figure 5.4: Decision boundaries of our neural models for the XOR problem. The blue shaded regions contain the points which the models will classify as 0, whereas the orange shaded region contains the points which the models will classify as 1. Since all the four instances corresponding to the XOR problem lie within their corresponding classification region, the problem is solved with an accuracy of 1. Left figure corresponds to zero-order hold, whereas the figure on the right corresponds to the dynamical source of input.

$w_2 = 8.64606573$, $b = 2.19796426$ and $\phi = -0.20237278$, as well as $\kappa = 10$, $v_{\text{peak}} = 0.7$, $\alpha = 0.001$, $S = 12$ and the dot-sinusoidal transformations introduced in Section 5.1.1.2 as hyperparameters. As such, Figure 5.5 contains the decision boundary of the model for that configuration. We find the boundary quite illustrative, as its complex geometric shape shows that it is not the typical boundary a logistic regressor would give upon applying feature expansion. Moreover, that result comes from the difficulty we have had to train the model to arrive at the solution. Although the neuron has enough capacity to solve the problem, it truly seems that optimising the parameters is challenging, even though loss curves show coherent training patterns. As commented earlier, we have not found any solution when employing the zero-order hold technique, as the neuron partitions the plane at most into parallel strips, which is not enough to solve this problem.

In any case, showing our model can solve such a complex nonlinear problem serves as proof of the capacity that biologically plausible neural models can offer to tackle problems, provided a correct set of tools is employed. We believe that expanding this to a network of neurons may contribute to stabilising training while providing additional capacity to solve some of the hardest problems, but that remains to be investigated. To end with, we must highlight that every other training pattern observed when solving the XOR problem has also been observed in this context. In general, although the training framework seems to be correct, the neuron needs different trials to actually reach a valid solution. Adjusting hyperparameters is relevant to obtain a more or less stable training curve, but it is not easy to

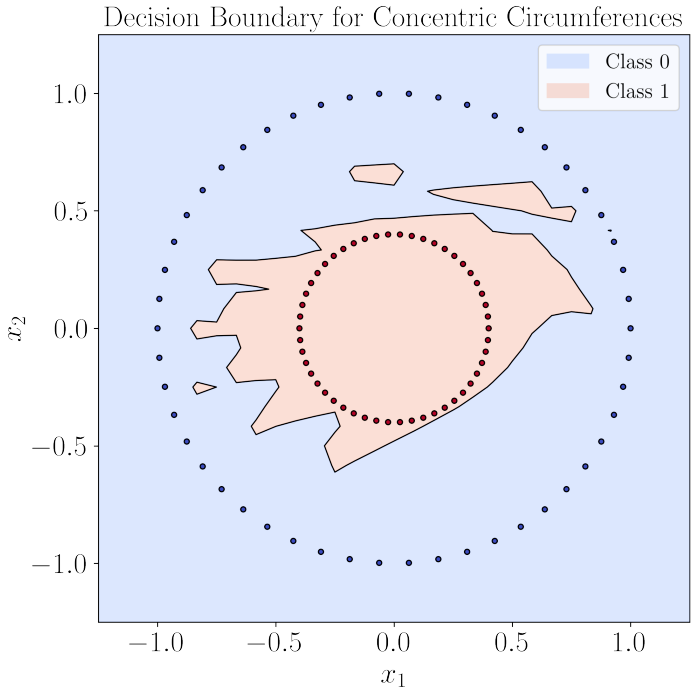

Figure 5.5: Decision boundary of our neural model for the concentric circumferences problem ($r_0 = 1$, $r_1 = 0.4$). The solution includes $w_1 = -5.7601033$, $w_2 = 8.64606573$, $b = 2.19796426$ and $\phi = -0.20237278$, as well as $\kappa = 10$, $v_{\text{peak}} = 0.7$, $\alpha = 0.001$, $S = 12$ and the dot-sinusoidal transformations introduced in Section 5.1.1.2 as hyperparameters. The model learns a curious boundary that effectively solves the problem but that it shows the difficulty we have had to train the model so as to arrive to that solution.

determine whether the neuron will arrive at a valid solution for the problem.

5.3.3 Effect of Hyperparameters and Comparison Against Common Implementations of Logistic Regressors

To end with, this subsection will cover the results we have collected when analysing the effect of hyperparameters on the training patterns of the neuron, as well as when comparing maximum accuracy, mean accuracy, training time per epoch and accuracy to training time ratio against common implementations of logistic regressors.

5.3.3.1 Effect of Hyperparameters on Training

We shall begin by commenting on the characteristics of the training procedure that we have observed. Leaving aside that zero-order hold provides a more stable training, both techniques show analogous training patterns when adjusting different hyperparameters. That is the

reason why the following results will be addressed on top of the zero-order hold modelling technique. Figure 5.6 shows several training patterns that arise when adjusting different hyperparameters. On the one hand, we have noted that the model trains following a typical gradient-descent-based scheme, where loss decays quickly in the first iterations but then stabilises as the parameter set approaches an optimum (which could either be a solution or not). However, note that the initial loss is very sensitive to how the parameters are initialised, as well as that not all trials result in a smooth loss curve. We think the optimisation procedure is especially challenging in our scenario, but results show that training is coherent, regardless of whether the model arrives at a solution. As for different configurations of hyperparameters, we have observed the following patterns:

- Increasing the learning rate from 0.001 up to 0.1 has a direct effect on the stability of the training procedure. Looking at the figures in the middle, for a sufficiently large value of α, the loss curve begins to show steep transitions from iteration to iteration. This was a property we expected, as a large learning rate produces faster convergence in the first iterations, but then destabilises training.

- The gain factor does not appear to have a significant impact on the results. As the figures on the bottom show, modifying κ does not produce noteworthy training patters. It is true that we have observed that the lower κ is, the lower the initial value of the loss is on average, but optimisation becomes more challenging.

- We have observed that a configuration of $\kappa = 10$ and $\alpha = 0.001$ results in reasonably stable training curves. However, stability is highly dependent on the initial values of the parameters.

5.3.3.2 Comparison of the Results against Common Implementations of Logistic Regressors

Finally, we have proposed a set of experiments that seeks to compare the capacity of our model against common implementations of logistic regressors. As noted in introducing the methodology of this chapter, we have trained logistic regressors, implemented in *scikit-learn*, *Keras* and *PyTorch*, on both problems we have discussed in order to compare their performance with our reformulation of the perceptron. Not only have we focused on the maximum accuracy each implementation offers, but also on other relevant aspects for training, such as the efficiency of the training procedure (the time it takes to complete one epoch), the accuracy to training time ratio, etc.

Let us begin by introducing the main results for both the XOR and the concentric circumferences problems. Tables 5.1 and 5.2 contain those, respectively, together with the exact parameterisation and conditions that we have contemplated. The results observed for both problems align with our initial expectations, as the regressors implemented in *scikit-learn*, *Keras* and *PyTorch* achieve a maximum accuracy of 0.75 in the XOR problem and a maximum of 0.68 in the concentric circumferences problem, whereas our models solve both problems

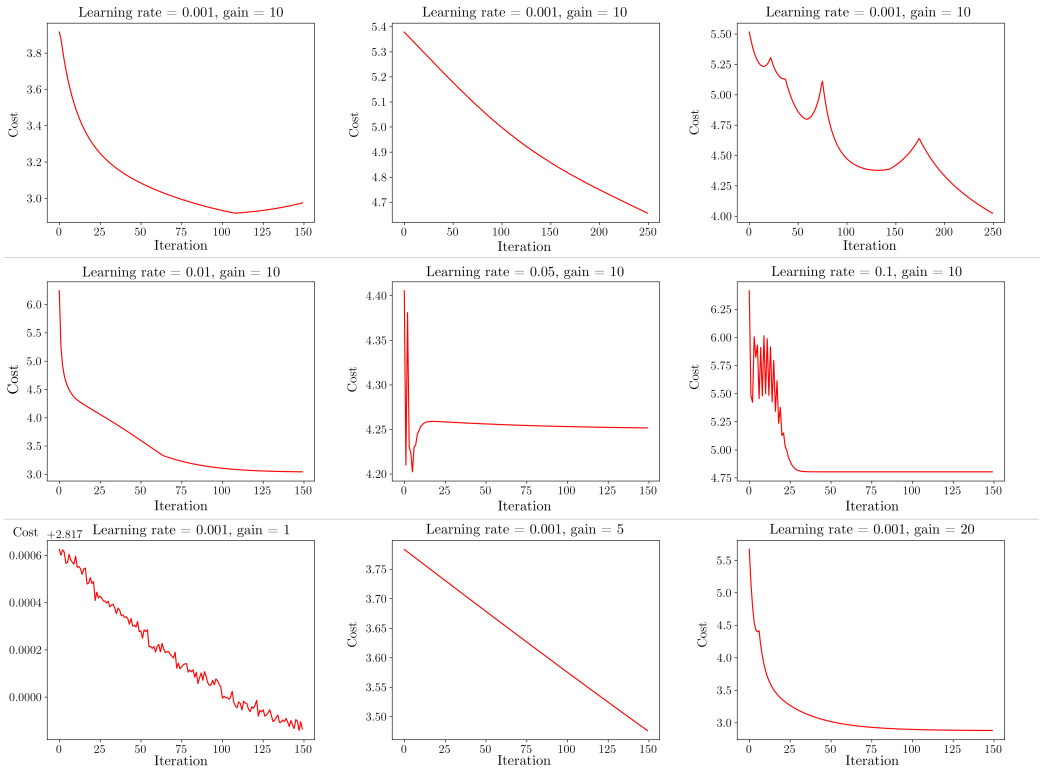

Figure 5.6: Training dynamics of our neuron for different configurations of hyperparameters. Top figures show three different trials for randomly initialised weights (following Xavier initialisation [75] for w_i and a uniform distribution for b and ϕ), $\kappa = 10$ and a learning rate of 0.001. Middle figures show other three trials for randomly initialised weights, $\kappa = 10$ and learning rates of 0.01, 0.05 and 0.1, respectively. Bottom figures show other three trials with randomly initialised weights, learning rate of 0.001 and $\kappa = 1, 5$ and 20, respectively. Every experiment has been conducted by setting $v_{\text{peak}} = 1$. All the trials correspond to the zero-order hold modelling technique.

perfectly (in the case of the concentric circumferences problem, only a valid solution is found when the dynamical source of input modelling technique is considered). However, among the 100 trials we have conducted, only a few have been successful employing our models, since their mean accuracy is considerably lower than the maximum one for both problems. This supports our hypothesis that training a single neuron to solve complex problems is quite challenging, and further research is needed to develop neural networks that may achieve superior performance with a more stable and robust training.

However, obtaining the highest accuracy score for our models is not the only focus we need to consider. For instance, it is clear that training an epoch of our models is much more resource-demanding than training any logistic regressor we have considered. We expected this, as our models need to integrate a differential equation per instance, irrespective of how data is inputted (via zero-order hold or via dynamical source of input). In addition, the backward pass also requires solving a differential equation per instance, and although the

Training Performace	Max Acc.	Mean Acc.	Time / Epoch (s)	Acc. / Time
scikit-learn	0.75	0.44	$\mathbf{6.31\times10^{-5}}$	**7012.70**
Keras	0.75	0.50	0.08	6.44
PyTorch	0.75	0.50	3.70×10^{-3}	135.20
QIF Perceptron ZOH (ours)	**1.00**	**0.63**	0.49	1.05
QIF Perceptron DSI (ours)	**1.00**	0.59	0.46	1.28

Table 5.1: Training performance comparison for logistic regressors implemented in *scikit-learn*, *Keras* and *PyTorch* and our neural model on the XOR problem. All models have been trained 100 times for 50 epochs with a batch size of 1 and $\alpha = 0.001$, and maximum accuracy, mean accuracy, training time per epoch (in seconds) and accuracy to training time ratio have been measured. As for the hyperparameters of our model, $\kappa = 10$ for both cases, $v_{\text{peak}} = 1$ for zero-order hold and $v_{\text{peak}} = 0.6$, $S = 3$ and the dot-sinusoidal transformations for dynamical source of input.

Training Performace	Max Acc.	Mean Acc.	Time / Epoch (s)	Acc. / Time
scikit-learn	0.68	0.50	$\mathbf{3.18\times10^{-5}}$	**15798.98**
Keras	0.50	0.50	0.11	4.57
PyTorch	0.68	0.51	0.02	31.45
QIF Perceptron ZOH (ours)	0.80	**0.65**	10.73	0.06
QIF Perceptron DSI (ours)	**1.00**	0.52	10.50	0.05

Table 5.2: Training performance comparison for logistic regressors implemented in *scikit-learn*, *Keras* and *PyTorch* and our neural model on the concentric circumferences problem. All models have been trained 100 times for 50 epochs with a batch size of 4 and $\alpha = 0.001$, and maximum accuracy, mean accuracy, training time per epoch (in seconds) and accuracy to training time ratio have been measured. As for the hyperparameters of our model, $\kappa = 10$ for both cases, $v_{\text{peak}} = 1$ for zero-order hold and $v_{\text{peak}} = 0.7$, $S = 12$ and the dot-sinusoidal transformations for dynamical source of input.

adjoint state method provides a constant-memory solution (with respect to the number of integration coefficients), for problems with many instances and a small network, training can become expensive compared with the efficiency the logistic regressors show. We think that the adjoint state method will really have a considerable impact when building neural networks with our neural model, since the backward pass will increase in complexity only if the number of instances to process grows, but not if the architecture gets considerably more complex. However, this is also a hypothesis that we will have to prove. All in all, if time is a priority and we do not care about obtaining perfect results, *scikit-learn* might conceivably be the best option (it achieves the highest accuracy-time ratio), but if we truly care about solving the problem, results show our models are good candidates, even if time and different trials will be required to obtain a valid solution.

This concludes the last structural chapter of this thesis. After formalising what we consider a biologically plausible neural representation, experimental validation has been conducted, and we consider that the results we have obtained for the aspects we have analysed are robust and consistent so as to highlight the remarkable capacity our approach offers. Training can be challenging and time-consuming, but it ultimately pays off. Before ending, we would like to mention that a complete implementation of our model is offered in the following repository in case the reader wants to use it: https://github.com/immp16/bioneurons

CHAPTER 6

Conclusions and Future Work

This has been a long journey where we have explored how biologically plausible representations of neural activity can contribute to the development of artificial intelligence in an alternative but compatible manner. The thesis has mainly focused on artificial neurons, which are the building blocks of deep learning, but similar ideas could be employed in other fields within artificial intelligence. This section aims to provide a summary of the thesis with the most relevant conclusions we have observed, as well as to comment on the research fields that have been devised. The modelling we have proposed and developed is certainly a building block of what a complete neural network could be, and therefore, there are hundreds of aspects that could be integrated in future research.

6.1 Conclusions

The hypothesis on which this thesis has been built relates the capacity of our neurons to acquire, process and respond to reality with current artificial intelligence that tries to mimic neural activity for machines to behave as we would. In fact, artificial neural networks only model some specific aspects of certain neurons and yet have achieved unprecedented performance on very complex tasks, such as language modelling. Therefore, should we increase the biological plausibility of those models by including other relevant aspects, could we truly provide machines with human-like capacities? Moreover, biological neurons process signals and manage resources more efficiently than artificial neurons; studying their behaviour could therefore help mitigate the efficiency limitations of current artificial neural networks. This thesis has tried to delve into answering those concerns and devising new horizons on which to work. In that journey, we have discovered several findings that we will summarise in this subsection.

- **Neural-based machine learning lacks biological plausibility.** After conducting a thorough and critical investigation of the computational properties of neurons, we

have concluded that unlike classic computational neuroscience, neural-based machine learning does not accurately model these aspects. For instance, not including the idea of time being a core component on top of which a neural model is built prevents typical deep learning models from behaving as real neural networks would. Bifurcations can not be inferred from that static paradigm, neither can resonator neurons be modelled. It is essential to consider different neurocomputational properties to enhance biological plausibility, something that remains largely unaddressed in modern machine learning.

- **Biological plausibility and effectiveness can be enhanced by introducing time and neurocomputational models into existing neural machine learning models.** The neural model we have worked on has tried to build upon the existing machine learning framework, but introducing some properties that enhance the biological plausibility of existing neural models within machine learning. Upon integrating time and the quadratic integrate-and-fire model into the modern perceptron, we have built a neuron that dynamically responds to the data being inputted, thus allowing it to employ its rich dynamics to enhance its capacity and solve more effectively some problems classic artificial neurons or logistic regressors struggle with.

- **Backpropagation helps mantain a balance between plausibility and efficiency.** Since one of our main goal has been to make this model compatible with existing machine learning frameworks, we have developed the theoretical formulation to train this model using the backpropagation algorithm. We are aware of the biological plausibility concerns regarding backpropagation, but it is an efficient and an effective procedure to train neural models, and therefore, suitable to maintain a balance between plausibility and efficiency.

- **The adjoint state method offers a constant-memory solution to compute the gradient of the loss.** Apart from deriving the analytical gradient, we have developed a constant-memory procedure via the adjoint state method to optimise learnable parameters without requiring a set of equations that scales with the number of learnable parameters; only a differential equation must be solved to backpropagate the error.

- **The model has enough capacity to solve some nonlinear problems, even though its optimisation is challenging.** Empirical results have validated the entire theoretical formulation we have provided in Chapter 4. Even though our main goal consisted of verifying that training followed a typical scheme and that the neuron acquired firing patterns in accordance with the class to be predicted, we tried to solve some nonlinear problems that classic neural models struggle with. In fact, no logistic regressor can solve the XOR or the concentric circumferences problem without requiring further additions, such as feature expansion or additional neurons, yet we have been able to successfully solve both problems employing our model. It is true that this has required analysing different procedures to transform features into coherent continuous signals, but our two proposals, the zero-order hold model and the dynamical source of input, have allowed us to achieve such results without needing to extend our neuron or expand the number of features (or similar procedures to enhance the capacity of the model).

All in all, we have found that biologically plausible representations may contribute to a robust modelling and development of artificial intelligence. While it is true that training our model has been challenging and finding valid solutions has required numerous trials, we are confident that research on this topic will bear fruit in the long term.

6.2 Future Work

Now that we have established the building block of an artificial neural network, we can proceed to incorporate all the suggestions, proposals and considerations that recent research on artificial intelligence, and especially on machine learning, has brought about. In fact, there are so many lines of research we could focus on that it would be virtually impossible to cover them all. However, in this subsection, we will offer some relevant lines of research that we have devised.

- **Analyse the actual capacity and the limitations of our neural model.** It is mandatory to understand which types of problems this neural model can solve, under which circumstances and the conditions under which the resolution will be effective and efficient. For instance, if the universal approximation theorem were somehow applicable to our model, we should analyse whether it would be a better idea to improve training dynamics on a single neuron or to work on neural networks to divide the workload and improve or stabilise training and inference.

- **Extend the model to a network of neurons.** In relation to the previous field, another line of research would consist of extending this model to a network of neurons. However, this is not an easy task, as deriving the gradient of the loss with respect to the learnable parameters requires further meticulous analysis on how error is backpropagated through the dynamics of each neuron. The adjoint state method can still be employed, but adding more neurons makes it difficult to derive exact formulas.

- **Research different optimisation methods or enhance dynamics.** Some alternatives to extending the model to neural networks would require either researching different optimisation methods that contain biologically plausible properties or enhancing the inner dynamics of each neuron employing n-dimensional differential equations. However, the former case requires a broad analysis on the feasibility of other training procedures, whereas the latter also requires adapting the gradient of the loss, as the coupling of dynamics has an effect (although not significant) on the adjoint state that requires adapting the formulas according to the dynamics considered for the neuron.

- **Voltage peak as a learnable parameter** Making v_{peak} a learnable parameter would be a natural extension to pursue in the near future. While the adjoint state will also suffer small adjustments, this is an easy problem that we can solve.

- **Research parameter initialisation, regularisation and implementation improvements.** Some other lines of research can focus on specific contributions to the whole

framework on which we have been working. For instance, parameter initialisation is crucial for the neuron to arrive at valid solutions during training. Researching feasible and effective alternatives would have a positive impact on the number of trials and the amount of time required to solve tasks. Regularisation techniques are also interesting in order to verify whether they contribute to a more stable training. In addition, providing paralelisation options, faster resolution of differential equations and automatic differentiation would contribute to a more efficient training. These are by no means trivial tasks, but they could help bridge the gap between biological efficiency and the performance gains we seek.

There is definitely work to be done, since this is a relatively new field of research. Spiking neural networks and neural ordinary differential equations have provided several sets of tools that have actually made it possible to conceive this hybrid reformulation of the perceptron adapted to the existing machine learning framework. While this thesis represents an initial milestone, we hope that it has helped shed light on how future artificial intelligence could be developed to live up to its name.

APPENDIX A

On Dynamical Systems in Neuroscience

A.1 Evolution of the State Variable of One-Dimensional Systems through the Time Derivative of F

This section will cover how the evolution of the state variable of a one-dimensional system can be determined through the time derivative of F, leading to attraction and repulsion domains, as well as to the effect of equilibria on the state variable. Suppose the time derivative of $F(V(t)) = \frac{dV}{dt}(t)$ has three roots, V_1, V_2 and V_3, where $V_1 < V_2 < V_3$, as in Figure 2.5 (top left). We shall consider the following cases for initial (or current) states and values of $F(V)$:

1. $V(0) < V_1$ and $F(V) < 0,\ \forall V$ for which $V < V_1$. Since the time derivative of V is negative, then $\forall \epsilon_1, \epsilon_2 > 0$, if $\epsilon_1 < \epsilon_2$, then $V(t+\epsilon_1) > V(t+\epsilon_2)$, and therefore, $\lim_{\epsilon \to \infty} V(t+\epsilon) = -\infty$ (since there is no other root). In other words, a negative time derivative of the state variable indicates that the state variable will decrease over time. The magnitude of the change will be given by the slope at the current state value.

2. $V(0) < V_1$ and $F(V) > 0,\ \forall V$ that satisfies $V < V_1$. Since the time derivative of V is positive, we have that $\forall \epsilon_1, \epsilon_2 > 0$, if $\epsilon_1 < \epsilon_2$, then $V(t+\epsilon_1) < V(t+\epsilon_2)$, and therefore, $\lim_{\epsilon \to \infty} V(t+\epsilon) = V_1$. The state variable will converge into V_1 asymptotically, because $F(V_1) = 0$ and because $F'(V)$ will have its magnitude decreased as it approaches V_1, thus the difference between $V(t+\epsilon_1)$ and $V(t+\epsilon_2)$ gradually becoming smaller. This does not imply that the state variable will always evolve in a smooth and monotonic way, since depending on the characteristics of F, the trajectory can oscillate around the resting state. In any case, once V reached its asymptotic value V_1, it would not be able to continue evolving. We therefore see that V_1 is an attractor, because for every

initial state $V(0) < V_1$, the state variable will be attracted to V_1 provided that $F(V)$ is positive for the state values in the attraction domain.

3. $V_1 < V(0) < V_2$ and $F(V) > 0, \forall V$ that satisfies $V_1 < V < V_2$. Since the time derivative of V is positive, we have that $\forall \epsilon_1, \epsilon_2 > 0$, if $\epsilon_1 < \epsilon_2$, then $V(t + \epsilon_1) < V(t + \epsilon_2)$, and therefore, $\lim_{\epsilon \to \infty} V(t + \epsilon) = V_2$. Recall that in the first case, V_1 was not an attractor, but since the time derivative is positive, it is V_2 that has become an attractor.

4. $V_1 < V(0) < V_2$ and $F(V) < 0, \forall V$ that satisfies $V_1 < V(0) < V_2$. Since the time derivative of V is negative, we have that $\forall \epsilon_1, \epsilon_2 > 0$, if $\epsilon_1 < \epsilon_2$, then $V(t + \epsilon_1) > V(t + \epsilon_2)$, and therefore, $\lim_{\epsilon \to \infty} V(t + \epsilon) = V_1$. Recall that in the second case, V_1 was an attractor, and this is further emphasised when approaching it from the right-hand side. As in the previous cases, the state value V_1 will be reached asymptotically, since only when $\epsilon_1, \epsilon_2 \to \infty$, $V(t + \epsilon_1) = V(t + \epsilon_2) = V_1$; in the meantime, the difference between both will gradually decrease as $F'(V)$ approaches 0.

We could continue to analyse the evolution of the state variable for the cases where $V_2 < V(0) < V_3$ and $V(0) > V_3$, but the patterns that arise are actually identical. We see that all state variables will either evolve to $\pm\infty$ or to some of the roots of $F(V)$.

A.2 Determining Types of Equilibria through the Jacobian Matrix of the Linearised System in Two-Dimensional Systems

This section will cover the full derivation of how types of equilibria (for hyperbolic ones) can be determined via the Jacobian matrix of the linearised system for two-dimensional systems. First of all, the solution of the linear system described in Equation (2.20) is given by Equation (A.1) (it is easy to check its validity by differentiating the solution):

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = e^{Jt} \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \sum_{n=0}^{\infty} \frac{(Jt)^n}{n!} \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}, \tag{A.1}$$

where J is the Jacobian matrix of the system and $(x(0), y(0))$ the initial solution of the system. Now, suppose J is diagonalisable, this is, suppose there exist $V \in GL_2(\mathbb{R})$ and $D = \text{diag}(\lambda_1, \lambda_2) \in M_2(\mathbb{R})$ such that $J = VDV^{-1}$, where λ_1 and λ_2 are the eigenvalues of J. By the properties of the matrix exponential, we have that

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = Ve^{Dt}V^{-1} \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

Considering V is constructed from the coordinates of the eigenvectors of J, v_1 and v_2, which are independent provided that $\lambda_1 \neq \lambda 2$ (as they span different eigenspaces in that case),

we shall write $(x(0), y(0)) = c_1 v_1 + c_2 v_2$, where $c_1 \neq 0$ or $c_2 \neq 0$. By substituting this expression into the matricial equation via the coordinates of the eigenvectors we have that

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = V e^{Dt} V^{-1} (c_1 M_{\beta_k}(v_1) + c_2 M_{\beta_k}(v_2)) + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

By developing the previous expression taking into account that $V^{-1} M_{\beta_k}(v_i) = e_i$, we have that

$$\begin{aligned} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} &= c_1 V e^{Dt} V^{-1} M_{\beta_k}(v_1) + c_2 V e^{Dt} V^{-1} M_{\beta_k}(v_2) + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ &= c_1 V e^{Dt} M_{\beta}(e_1) + c_2 V e^{Dt} M_{\beta}(e_2) + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}. \end{aligned}$$

To end with, since $D = \operatorname{diag}(\lambda_1, \lambda_2)$ is diagonal, we can use the property that $e^{Dt} = \operatorname{diag}(e^{\lambda_1 t}, e^{\lambda_2 t})$. Therefore, the previous expression can be developed as follows:

$$\begin{aligned} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} &= c_1 V \begin{pmatrix} e^{\lambda_1 t} & 0 \\ 0 & e^{\lambda_2 t} \end{pmatrix} M_{\beta}(e_1) + c_2 V \begin{pmatrix} e^{\lambda_1 t} & 0 \\ 0 & e^{\lambda_2 t} \end{pmatrix} M_{\beta}(e_2) + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ &= c_1 V \begin{pmatrix} e^{\lambda_1 t} \\ 0 \end{pmatrix} + c_2 V \begin{pmatrix} 0 \\ e^{\lambda_2 t} \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ &= c_1 \begin{pmatrix} M_{\beta_k}(v_1) & M_{\beta_k}(v_2) \end{pmatrix} \begin{pmatrix} e^{\lambda_1 t} \\ 0 \end{pmatrix} + c_2 \begin{pmatrix} M_{\beta_k}(v_1) & M_{\beta_k}(v_2) \end{pmatrix} \begin{pmatrix} 0 \\ e^{\lambda_2 t} \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ &= c_1 e^{\lambda_1 t} M_{\beta_k}(v_1) + c_2 e^{\lambda_2 t} M_{\beta_k}(v_2) + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}. \end{aligned}$$

This matrix equation can be vectorially written to provide a general solution to the linearised system, as Equation (A.2) shows:

$$(x(t), y(t)) = c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 + (x_0, y_0). \tag{A.2}$$

However, how can Equation (A.2) unveil the nature of the resting point (x_0, y_0)? Note that if $\lambda_1, \lambda_2 < 0$, then

$$\lim_{t \to \infty} (x(t), y(t)) = \lim_{t \to \infty} [c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 + (x_0, y_0)] = (x_0, y_0),$$

effectively confirming (x_0, y_0) is an asymptotically (and also exponentially) stable node. Likewise, if $\lambda_1, \lambda_2 > 0$, we have that trajectories diverge from the neighbourhood of (x_0, y_0), effectively confirming that it is an unstable node. It is also not difficult to infer that for eigenvalues with distinct signs, trajectories following the direction the eigenvector corresponding to the negative eigenvalue points to will converge into (x_0, y_0), whereas trajectories following the direction the eigenvector corresponding to the positive eigenvalue points to will diverge from it, effectively confirming that (x_0, y_0) is a saddle. In summary, types of equilibria can be determined by looking at the signs of the eigenvalues of the Jacobian matrix associated with the system.

There are still some cases we need to address, as both foci and neutrally stable equilibria can not be inferred from what we have seen. To begin with, foci are related to complex eigenvalues of the Jacobian matrix. In case J has complex eigenvalues $\lambda_{1,2} = a \pm ib$ (they will necessarily be complex conjugates, as well as their corresponding eigenvectors), we can not diagonalise it over $\mathbb{R}$, but we can still try to see if J is diagonalisable over $\mathbb{C}$. In such a case, Equation (A.2) is still valid, since $\mathbb{C}^2$ being a $\mathbb{C}$-vector space, the same rules can be applied, but it takes a specific form considering that the eigenvalues are complex:

$$\begin{aligned}(x(t), y(t)) &= c_1 e^{(a+ib)t} v + c_2 e^{(a-ib)t} \bar{v} + (x_0, y_0) = e^{at}(c_1 e^{ibt} v + c_2 e^{-ibt} \bar{v}) + (x_0, y_0) \\ &= e^{at}[c_1(\cos(bt) + i\sin(bt))v + c_2(\cos(bt) - i\sin(bt))\bar{v}] + (x_0, y_0).\end{aligned}$$

Note that for $(x(0), y(0))$ to be a linear combination of two complex eigenvectors, c_1 and c_2 must also be complex conjugates. Therefore, we have that

$$\begin{aligned}(x(t), y(t)) &= e^{at}[c(\cos(bt) + i\sin(bt))v + \bar{c}(\cos(bt) - i\sin(bt))\bar{v}] + (x_0, y_0) \\ &= e^{at}(\mathrm{Re}(2cv)\cos(bt) + \mathrm{Im}(-2cv)\sin(bt)) + (x_0, y_0).\end{aligned}$$

This expression shows that trajectories will follow an oscillatory path, revealing the existence of foci. In fact, if $a < 0$, then we have that

$$\lim_{t\to\infty}(x(t), y(t)) = \lim_{t\to\infty}[e^{at}(\mathrm{Re}(2cv)\cos(bt) + \mathrm{Im}(-2cv)\sin(bt)) + (x_0, y_0)] = (x_0, y_0),$$

effectively confirming spirals will converge into (x_0, y_0), thus the equilibrium point being a stable focus. Likewise, if $a > 0$, spirals will diverge from the resting state, thus revealing that it is an unstable focus.

Bibliography

[1] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958, doi: 10.1037/h0042519. See pages 1, 52, and 53.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, Dec. 1943, doi: 10.1007/BF02478259. See page 2.

[3] A. Vaswani *et al.*, "Attention is all you need," in *Adv. Neural Inf. Process. Syst.*, vol. 30, Dec. 2017. See pages 2, 56.

[4] T. Brown *et al.*, "Language models are few-shot learners," in *Adv. Neural Inf. Process. Syst.*, Dec. 6-12 2020, pp. 1877–1901. See page 2.

[5] R. Rombach *et al.*, "High-resolution image synthesis with latent diffusion models," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 19-24 2022, pp. 10 684–10 695. See page 2.

[6] E. M. Izhikevich, *Dynamical Systems in Neuroscience.* USA: MIT Press, 2007. See pages 7, 11, 21, 24, 25, 32, 37, 38, 40, 45, 46, and 50.

[7] I. B. Levitan and L. K. Kaczmarek, *The Neuron: Cell and Molecular Biology.* New York, NY, USA: Oxford University Press, 2002. See pages 8, 9.

[8] C. Henley, *Foundations of Neuroscience.* East Lansing, MI, USA: Michigan State University, 2021. See page 9.

[9] M. Foster, *A Text-Book of Physiology pt. 3.* New York, NY, USA: McMillan & Company, 1897. See page 9.

[10] M. V. L. Bennett, "Physiology of electrotonic junctions," *Ann. N. Y. Acad. Sci.*, vol. 137, no. 2, pp. 509–539, Jul. 1966, doi: 10.1111/j.1749-6632.1966.tb50178.x. See page 9.

[11] D. Purves *et al.*, *Neurosciences.* De Boeck Supérieur, 2019. See page 10.

[12] C. Giménez, "Composition and structure of the neuronal membrane: molecular basis of its physiology and pathology," *Rev. Neurol.*, vol. 26, no. 150, pp. 232–239, Feb. 1998. See page 10.

[13] L. Mejlbro, "The complete solution of fick's second law of diffusion with time-dependent diffusion coefficient and surface concentration," in *Compl. Solut. Fick's 2nd Law Diffus. Time-Dep. Diffus. Coeff. Surf. Conc.*, 1996, pp. 127–158. See page 12.

[14] B. Hille *et al.*, *Ion Channels of Excitable Membranes.* Sunderland, MA, USA: Sinauer Associates, 2001. See page 14.

[15] E. R. Kandel *et al.*, *Principles of Neural Science.* New York, NY, USA: McGraw-Hill, 2000. See page 14.

[16] F. R. Quintela *et al.*, "A general approach to Kirchhoff's Laws," *IEEE Trans. Educ.*, vol. 52, no. 2, pp. 273–278, May 2009, doi: 10.1109/TE.2008.928189. See page 16.

[17] J. V. Halliwell, "Voltage clamp techniques," in *Microelectrode Techniques: The Plymouth Workshop Handbook*. The Company of Biologists Ltd, 1994, pp. 17–35. See page 16.

[18] J. C. Wester and D. Contreras, "Biophysical mechanism of spike threshold dependence on the rate of rise of the membrane potential by sodium channel inactivation or subthreshold axonal potassium current," *J. Comput. Neurosci.*, vol. 35, pp. 1–17, Aug. 2013, doi: 10.1007/s10827-012-0436-2. See page 19.

[19] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, Aug. 1952, doi: 10.1113/jphysiol.1952.sp004764. See page 20.

[20] M. Ito and T. Oshima, "Temporal summation of after-hyperpolarization following a motoneurone spike," *Nature*, vol. 195, no. 4844, pp. 910–911, Sep. 1962, doi: 10.1038/195910a0. See page 21.

[21] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer Science & Business Media, 2013. See page 27.

[22] D. M. Grobman, "Topological equivalence in the large for systems of differential equations," *Math. USSR–Sb.*, vol. 2, no. 4, pp. 535–542, Aug. 1967, doi: 10.1070/SM1967v002n04ABEH002353. See page 27.

[23] Y. A. Kuznetsov, *Elements of Applied Bifurcation Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1998. See pages 28, 51.

[24] E. Fehlberg, "Classical fourth-and lower order Runge-Kutta formulas with stepsize control and their application to heat transfer problems," *Computing*, vol. 6, pp. 61–71, Mar. 1970, doi: 10.1007/BF02241732. See pages 30, 86.

[25] H. Shang *et al.*, "Dynamical analysis of a stochastic neuron spiking activity in the biological experiment and its simulation by INa,P + IK Model," in *Adv. Neural Netw. – ISNN 2018*, 2018, pp. 850–859, doi: 10.1007/978-3-319-92537-0_96. See page 31.

[26] A. M. Lyapunov, "The general problem of the stability of motion," *Int. J. Control*, vol. 55, no. 3, pp. 531–534, Jun. 1992, doi: 10.1080/00207179208934253. See page 34.

[27] A. A. Andronov, *Theory of Bifurcations of Dynamic Systems on a Plane*. Israel Program for Scientific Translations, 1971. See page 37.

[28] Y. A. Kuznetsov, "Numerical analysis of bifurcations," in *Elements of applied bifurcation theory*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 505–585. See page 37.

[29] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophys. J.*, vol. 1, no. 6, pp. 445–466, Jul. 1961, doi: 10.1016/S0006-3495(61)86902-6. See page 37.

[30] J. Nagumo *et al.*, "An active pulse transmission line simulating nerve axon," *Proc. IRE*, vol. 50, no. 10, pp. 2061–2070, Oct. 1962, doi: 10.1109/JRPROC.1962.288235. See page 37.

[31] E. A. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*. New York, NY, USA: McGraw-Hill, 1955. See page 38.

[32] H. Hopf, "Vektorfelder in n-dimensionalen mannigfaltigkeiten," *Math. Ann.*, vol. 96, pp. 225–250, Dec. 1927, doi: 10.1007/BF01209164. See page 38.

[33] J. E. Marsden and M. McCracken, *The Hopf Bifurcation and its Applications*. Springer Science & Business Media, 1976. See page 40.

[34] L. Lapicque, "Recherches quantitatives sur l'excitation electrique des nerfs," *J. Physiol*, vol. 9, pp. 620–635, 1907. See page 44.

[35] R. B. Stein and A. L. Hodgkin, "The frequency of nerve action potentials generated by applied currents," *Proc. R. Soc. Lond. Ser. B Biol. Sci.*, vol. 167, no. 1006, pp. 64–86, Jan. 1967, doi: 10.1098/rspb.1967.0013. See page 44.

[36] H. C. Tuckwell, "Cortical network modeling: Analytical methods for firing rates and some properties of networks of lif neurons," *J. Physiol. Paris*, vol. 100, no. 1-3, pp. 88–99, Oct. 2006, doi: 10.1016/j.jphysparis.2006.09.001. See page 44.

[37] E. Shlizerman and P. Holmes, "Neural dynamics, bifurcations, and firing rates in a quadratic integrate-and-fire model with a recovery variable. i: Deterministic behavior," *Neural Comput.*, vol. 24, no. 8, pp. 2078–2118, Apr. 2012, doi: 10.1162/NECO_a_00308. See page 46.

[38] S. Ghilardi, "An algebraic theory of normal forms," *Ann. Pure Appl. Log.*, vol. 71, no. 3, pp. 189–245, Feb. 1995, doi: 10.1016/0168-0072(93)E0084-2. See page 46.

[39] E. M. Izhikevich, "Resonate-and-fire neurons," *Neural networks*, vol. 14, no. 6-7, pp. 883–894, Jul. 2001, doi: 10.1016/S0893-6080(01)00078-8. See page 48.

[40] G. Young, "Note on excitation theories," *Psychometrika*, vol. 2, no. 2, pp. 103–106, Jun. 1937, doi: 10.1007/BF02288064. See page 48.

[41] M. Haragus and G. Iooss, *Local Bifurcations, Center Manifolds, and Normal Forms in Infinite-Dimensional Dynamical Systems*. Springer Science & Business Media, 2011. See page 51.

[42] S. L. Brunton *et al.*, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 113, no. 15, pp. 3932–3937, Mar. 2016, doi: 10.1073/pnas.151738411. See page 51.

[43] R. Chen *et al.*, "Neural ordinary differential equations," in *Adv. Neural Inf. Process. Syst.*, vol. 31, Dec. 2018. See pages 51, 60, 61, and 79.

[44] C. Banerjee *et al.*, "An empirical study on generalizations of the relu activation function," in *Proc. 2019 ACM Southeast Conf.*, Apr. 2019, pp. 164–167. See page 53.

[45] Z. Zhang *et al.*, "Enhancing deep learning models for image classification using hybrid activation functions," Nov. 2023, doi: 10.21203/rs.3.rs-3574353/v1. See page 53.

[46] M. Lee, "Mathematical analysis and performance evaluation of the gelu activation function in deep learning," *J. Math.*, vol. 2023, no. 1, p. 4229924, Aug. 2023, doi: 10.1155/2023/4229924. See page 53.

[47] B. Singh *et al.*, "Analyzing the impact of activation functions on the performance of the data-driven gait model," *Results Eng.*, vol. 18, p. 101029, Jun. 2023, doi: 10.1016/j.rineng.2023.101029. See page 53.

[48] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, Jan. 1962, doi: 10.1113/jphysiol.1962.sp006837. See page 54.

[49] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980, doi: 10.1007/BF00344251. See pages 54, 55.

[50] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Adv. Neural Inf. Process. Syst.*, vol. 25, Dec. 2012. See page 55.

[51] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778. See page 55.

[52] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735. See page 55.

[53] R. Costa *et al.*, "Cortical microcircuits as gated-recurrent neural networks," in *Adv. Neural Inf. Process. Syst.*, vol. 30, Dec. 2017. See page 55.

[54] D. Bahdanau *et al.*, "Neural machine translation by jointly learning to align and translate," arXiv preprint arXiv:1409.0473, Sep. 2014, doi: 10.48550/arXiv.1409.0473. See page 56.

[55] L. Kozachkov *et al.*, "Building transformers from neurons and astrocytes," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 120, no. 34, p. e2219150120, Aug. 2023, doi: 10.1073/pnas.2219150120. See page 56.

[56] D. Krotov and J. Hopfield, "Large associative memory problem in neurobiology and machine learning," in *Int. Conf. Learn. Represent.*, Jan. 2021. See page 56.

[57] S. Sanaullah *et al.*, "Exploring spiking neural networks: a comprehensive analysis of mathematical models and applications," *Front. Comput. Neurosci.*, vol. 17, p. 1215824, Aug. 2023, doi: 10.3389/fncom.2023.1215824. See pages 59, 61.

[58] A. Cristini *et al.*, "A continuous-time spiking neural network paradigm," in *Adv. Neural Netw.: Comput. Theor. Issues*. Springer International Publishing, 2015, pp. 49–60. See page 61.

[59] T. C. Wunderlich and C. Pehle, "Event-based backpropagation can compute exact gradients for spiking neural networks," *Sci. Rep.*, vol. 11, no. 1, p. 12829, Jun. 2021, doi: 10.1038/s41598-021-91786-z. See pages 61, 72, and 79.

[60] D. E. Rumelhart *et al.*, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, May 1986, doi: 10.1038/323533a0. See page 62.

[61] A. Shrestha *et al.*, "Approximating back-propagation for a biologically plausible local learning rule in spiking neural networks," in *Proc. Int. Conf. Neuromorph. Syst.*, Jul. 2019, pp. 1–8. See page 63.

[62] J. Campbell *et al.*, "Considerations of biological plausibility in deep learning," *Cornell Undergrad. Res. J.*, vol. 1, no. 1, pp. 4–12, Apr. 2022, doi: 10.37513/curj.v1i1.660. See page 63.

[63] R. Gâteaux, "Sur les fonctionnelles continues et les fonctionnelles analytiques," *Compt. Rendus Acad. Sci. Paris*, vol. 157, no. 325–327, 1913. See page 67.

[64] W. Greiner *et al.*, *Path Integrals in Field Theory*. Springer-Verlag Berlin Heidelberg, 1996. See page 67.

[65] N. Pollini *et al.*, "Adjoint sensitivity analysis and optimization of hysteretic dynamic systems with nonlinear viscous dampers," *Struct. Multidiscip. Optim.*, vol. 57, pp. 2273–2289, Nov. 2018, doi: 10.1007/s00158-017-1858-2. See page 72.

[66] O. P. Agrawal, "Formulation of Euler–Lagrange equations for fractional variational problems," *Journal of Mathematical Analysis and Applications*, vol. 272, no. 1, pp. 368–379, Aug. 2002, doi: 10.1016/S0022-247X(02)00180-4. See page 73.

[67] D. Liberzon, *Calculus of Variations and Optimal Control Theory: a Concise Introduction*. Princeton, NJ, USA: Princeton Univ. Press, 2011. See page 73.

[68] L. S. Pontryagin *et al.*, *Mathematical Theory of Optimal Processes*. New York, NY, USA: Interscience Publ. & John Wiley & Sons, Inc., 1962. See page 79.

[69] R. Chen *et al.*, "Learning neural event functions for ordinary differential equations," in *Int. Conf. Learn. Represent.*, Jan. 2021. See page 79.

[70] L. Euler, *Institutiones Calculi Integralis*. Acad. Imp. Scient., 1792. See page 85.

[71] P. Lakrisenko *et al.*, "Efficient computation of adjoint sensitivities at steady-state in ode models of biochemical reaction networks," *PLOS Comput. Biol.*, vol. 19, no. 1, p. e1010783, Jan. 2023, doi: 10.1371/journal.pcbi.1010783. See page 88.

[72] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, Oct. 2011, doi: 10.5555/1953048.2078195. See page 91.

[73] F. Chollet *et al.*, "Keras," https://keras.io, 2015. See page 91.

[74] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Adv. Neural Inf. Process. Syst.*, vol. 32, Dec. 2019. See page 91.

[75] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Stat.*, vol. 9, May 2010, pp. 249–256. See pages 94, 98.