Design and Performance Analysis of a Neuromodulable Spiking Recurrent Cell in the Context of Supervised Learning

Auteur : Bernard, Aurélien
Promoteur(s) : Drion, Guillaume
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil électricien, à finalité spécialisée en "signal processing and intelligent robotics"
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/18193

UNIVERSITY OF LIÈGE - FACULTY OF APPLIED SCIENCES

SUPERVISED BY PROFESSOR GUILLAUME DRION & FLORENT DE GEETER

Design and Performance Analysis of a Neuromodulable Spiking Recurrent Cell in the Context of Supervised Learning

MASTER THESIS COMPLETED FOR THE PURPOSE OF OBTAINING THE MASTER'S DEGREE IN ELECTRICAL ENGINEERING BY

BERNARD Aurélien

Academic Year 2022-2023

Contents

Acknowledgements

I would like to take this opportunity to thank everyone who has helped and enabled me during my master's and this thesis. In particular, I would like to thank Professor Drion and Florent De Geeter for their patience, always finding time to give me valuable input, and giving me the opportunity to work on this project with them. Not to mention that Florent's SRC code was very well designed and helped me advance much faster in this project than if I had to build it all from scratch. In this thesis, I am lucky to have worked at the intersection of neuroscience and machine learning which have been two of my main interests since the beginning of my master's.

I would also like to thank Professor Vanderbemden and the other electrical engineering members of the jury that allowed me to join the master's in electrical engineering while having a computer science background. You have helped accomplish one of the decisions for which I am the most happy and proud to this day.

Finally, I could not write acknowledgements without mentioning my old(er) but little sister, DOCTOR (yes, in big letters) Adélaïde Bernard. Even if we could not see eye to eye when were kids, you have become one of my greatest sources of motivation and inspiration. You have helped me in more ways than you will ever know. Thank you, I could not have wished for a better sister.

Abstract

From the desire to understand how neurons work and how we learn, neuroscientists from early 1900 to the 1950s proposed increasingly more complex neuron models. However, no learning rule seemed to have a satisfactory performance until the invention of back-propagation which uses optimisation and calculus to optimise the performance at a certain task. This new learning rule required neural networks to steer away from highly non-linear biological neuron models to ones with continuous properties that allow them to work. Machine Learning researchers favoured this new technique and have obtained increasingly better results with increasingly larger networks, large amounts of data, more computing power necessary to use them, and inevitably, higher power requirements. In an attempt to solve the shortcomings of modern artificial neural networks, the machine learning community is looking again into bio-inspired spiking neural networks and new learning rules that would perform as well as backpropagation. To this date, these attempts have had varying levels of success but have all fallen short of the modern standards of machine learning. However, a recent contribution has proposed a new way of creating and training spiking neural networks by re-designing classical recurrent neural network cells to force them to exhibit spiking behaviour. This allows the use of backpropagation without the need for approximations or adaptations as other spiking neural network training methods have proposed in the past. In this work, we analyse the proposed spiking recurrent cell from a nonlinear systems dynamics perspective and observe some of its shortcomings. Then, we propose two new cells that extend and reduce some of the previously observed issues. Namely, we extend the cell by introducing neuromodulation capabilities to have 3 distinct types of excitability through the addition of a single parameter, and with the second proposed cell, we extend further its capabilities by introducing bursting behaviour. Finally, we perform tests on small neural networks composed of 42 neurons based on the former cell on the MNIST benchmark coded in spikes through latency, and rate coding. In the experiments, we first modulate the neurons to fixed firing types and analyse the performance and behaviour of the network on the 2 tasks. We then randomly initialise the modulation through the network and set it as a learnable parameter for the same tasks. In these tests, we have achieved 91% accuracy which is close to state-of-the-art performance on SNNs that contain more than 100 times the number of neurons. Furthermore, we have observed that in the current setup, the network does not use a heterogeneity of firing types in the network to obtain the best results. Instead, it converges to one of the newly introduced firing types.

Concisely, the main goal of this work is to introduce neuromodulation to a spiking neural network cell that learns through direct backpropagation, and show that modulation can also be learned through this same learning algorithm to obtain satisfactory results.

Chapter 1

Introduction

Since the first neural network model was proposed by McCulloch & Pitts in 1943 [1], it would be an understatement to say that machine learning and neuron models have improved. At the time of writing, a large language model has been able to pass the bar exam scoring in the 90th percentile of real exam tackers [2]. However, the field of machine learning that started by finding inspiration in biological brains [1] has diverged since the advent of automatic differentiation. It has shifted towards Artificial Neural Network(ANN) models that show pragmatic and useful mathematical properties leaving behind bio-plausibility and the inspiration in biologically intelligent agents. Even though this approach has shown performances that surpass humans in specific tasks it has also shown several limitations that are inherent to its paradigm.

In 2016 AlphaGo [3], a program powered by ANNs beats Lee Sedol, the world's best Go player at the time. This was thought impossible due to the highly abstract nature and combinatorial number of possible actions at every turn. However, AlphaGo lost on several aspects compared to its human adversary. The first is that AlphaGo was able to play but not to see or move its own pieces. Indeed, current AI tends to have incredibly narrow scopes as they are specifically trained to optimise a score. More importantly, current AI systems have an energy consumption of several orders of magnitude greater than a human's performing the same task. A human consumes around 100W of energy, of which 20W is used by the brain[4]. AlphaGo ran on several servers that consumed approximately 1.5 MW of power during the encounter. Not to mention that these 1.5 MW only take into account the energy consumption during a game, the training process of ANNs is also highly energy-demanding.

The high energy consumption of computers compared to a brain is mainly due to the time sparse nature of neural communication compared to digital electronics. Neurons mainly communicate through sparse and short *all or none* pulses referred to as *"spikes"*. Communication through discrete pulses makes it low energy consuming as well as resilient to noise. Biological neurons use spike timings and frequencies to transmit information[5]. On the other hand, digital computers rely on transistors for their signal processing. Transistors require constant power levels to communicate between them. Furthermore, to be resilient to noise, computers have to use high voltages relative to neurons for the transistors' output not to be affected.

The spike-generating process is a well-studied process that can be simulated using models with varying levels of complexity, from simple *Integrate and Fire*(IF)[6] neurons introduced in the early 1900s to precise conductance-based models such as the *Hodgking-Huxley* (HH) model [7] proposed in 1952. However, using these models for machine learning applications present various challenges. IF networks are computationally efficient but are not able to capture common behaviours seen in neurons, like different excitability types and the ability to switch between them. On the other hand, sophisticated conductance-based neuron networks allow to recreate many known firing patterns but their high complexity makes large-scale simulations unfeasible. Furthermore, all spiking neuron models share a single limiting factor which is the lack of knowledge about the learning process in biological brains. The precise mechanisms involved in how

we learn have yet to be discovered. This is reflected in the abundance of proposed learning algorithms in *Spiking Neural Networks* (SNN)[8] with limited performance and a lack of well-formalised ideas akin to Back Propagation (BP) [9] in current ANNs.

In this thesis, we explore a technique that allows the use of the well-formalised BP algorithm on a spiking neural network. To do so, we use the idea of using an ANN with feedback connections more commonly referred to as Recurrent Neural Networks(RNN) to modify the Spiking Recurrent Cell(SRC) and create the *Modulable Spiking Recurrent Cell*(mSRC). The mSRC is able to spike like the SRC but adds the ability to switch between 3 different spiking modes. Furthermore, the neuromodulation capability is regulated by a single parameter that can be learned through backpropagation which allows the network to learn the optimal firing mode at the neuron level.

This thesis will be organised as follows:

- We start by laying the basis of neuron modelling and current machine learning techniques.
- Then, we formalise how the parallel between a feedback system and a recurrent neural network was used to analyse the SRC and develop its modulable mSRC extension and its bursting extension.
- We continue by analysing the proposed mSRC in order to show that it is able to capture 3 different spiking behaviours that can be modulated with a single parameter.
- Next, we implement the mSRC in a machine-learning environment and assess its performance on a latency and rate-coded image classification task.
- We finish by discussing the limitations of the technique as well as avenues to improve and expand this idea.

Chapter 2

Modelling Neuronal Excitability

Neurons are cells that constitute the basic building block of the nervous system. We can find them in the brain but also in the spinal cord, or attached to other tissues. Neurons are the main cells responsible for carrying and processing signals all around the body. Transmission between each other is done through their connections referred to as *synapses*. Neurotransmitters made up of chemicals and ions, are the vehicle that crosses synapses between neurons to establish electrochemical connections between them.

Figure 2.1 shows the structure of a neuron with its 3 main parts:

- **Dendrites** are the receiving end of neurons. They collect incoming neurotransmitters from the "*pre-synaptic*" or sender neurons. They integrate the inputs and transmit them to the body of the cell as graded potentials.
- **The soma** contains the nucleus of the cell. Its main function from a signal processing perspective is to integrate the graded potentials as they come from dendrites and generate short electrical impulses referred to as *spikes* or *action potentials*.
- **The axon** transmits the action potentials generated by the soma over long distances to post-synaptic neurons or other tissues. At the end of axons, we can find synaptic vesicles full of neurotransmitters that are opened when the action potential is received. A simplified depiction of an axons terminal can be seen in figure 2.2

From the description of a neuron alone, we can already see the basics of neuron communication and information processing. Communication starts at the pre-synaptic neuron where a spike is generated at the soma and transmitted down the axon. The axon then releases neurotransmitters from its vesicles that cross the synapse gap and get to the dendrites of the postsynaptic neuron. There, dendrites receive the neurotransmitter and create small graded electrical potentials that are transmitted to the soma of the postsynaptic neuron. Finally, the postsynaptic soma integrates these potentials coming from different synapses through time and decides if it should, or not, send a spike along its axons.

How does the soma make this decision? This is one of the main basis of computation in the brain. In this chapter, we aim at giving a deeper mathematical framework from which the operations performed by neurons can be formalised, better understood, and more importantly, replicated.

2.1 Biophysical Neuron Models

The working principles of neurons are simple however, the complexity lies in the spike generation mechanism. The main stages that are observed in a spike can be seen in figure 2.3. Spikes can be generated as a result of specific input intensities, their temporal pattern, and their spatial patterns. Furthermore, as figure

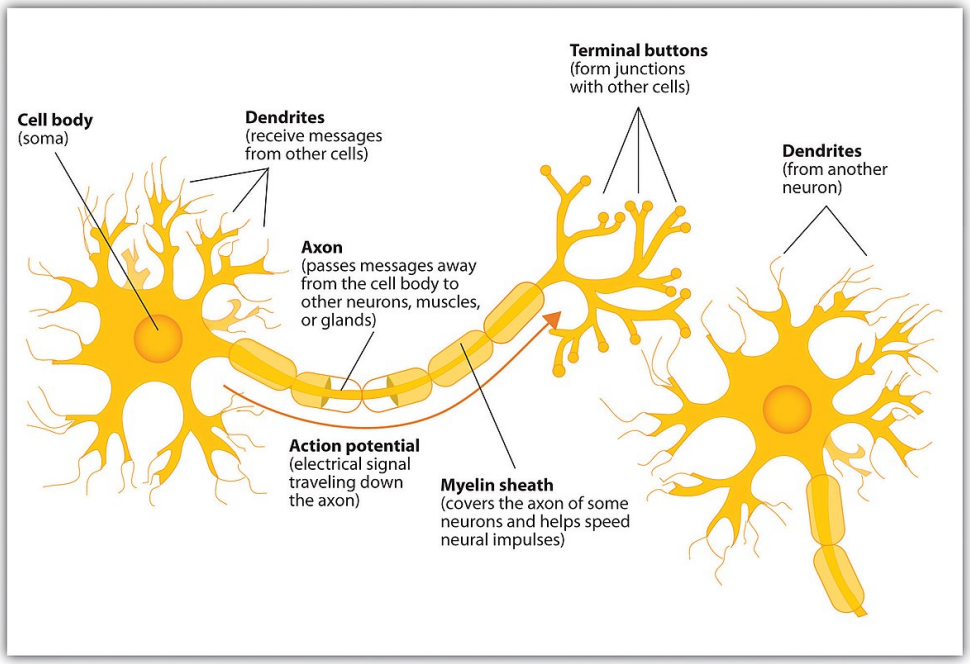


Figure 2.1: Simplified diagram of a neuron's anatomy. Taken from source.

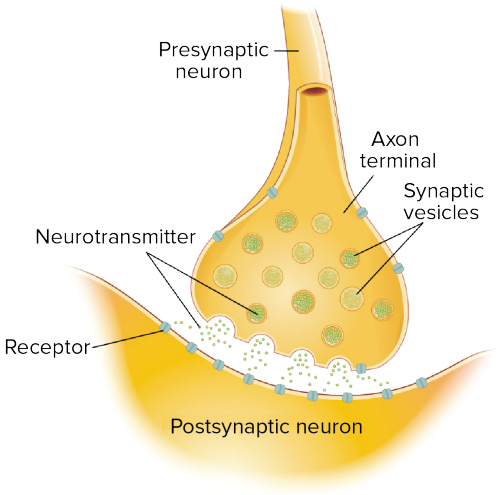


Figure 2.2: Simplified diagram of a neuron's axon terminal anatomy. Adapted from source.

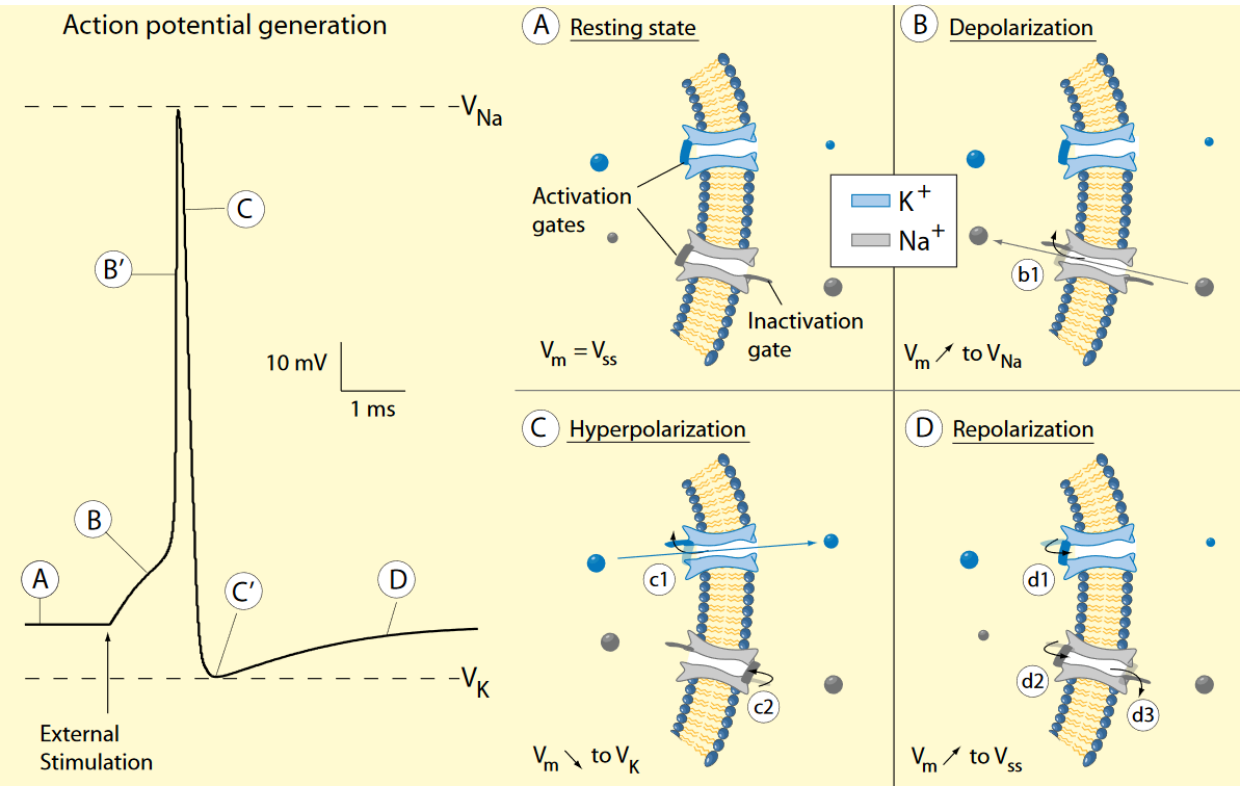


Figure 2.3: Description of the main stages in generating an action potential taken from [10]. During stage A the dynamics of the system are balanced. In B the equilibrium is disturbed by an input current and the membrane potential increases steadily. With sufficient input, stage B will evolve into B', a rapid acceleration of the membrane depolarisation creating the 'Upstroke' of the spike. After reaching a maximum voltage, in C, the neuron returns to its resting state by overshooting the equilibrium which is known as hyperpolarisation. Finally, a slow repolarisation(D) returns the membrane potential to its equilibrium.

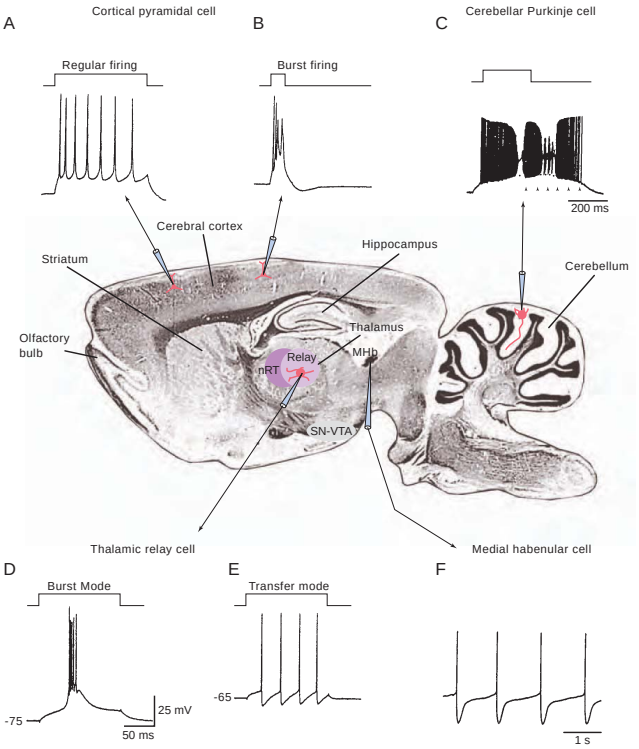


Figure 2.4: Mammalian brain neurons exhibit widely varying electrophysiological properties in response to a current step depending on sampled areas of the brain. Figure taken from [11].

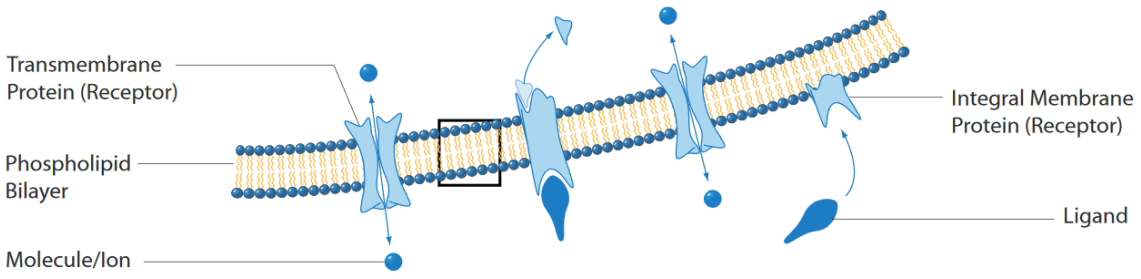


Figure 2.5: Simplified diagram of ion channels on dendrites. Adapted from [10].

2.4 shows, the shape and intensity of the generated spikes can also be widely different from one neuron to another. To recreate these behaviours, a large variety of models have been proposed, from the simple Integrate and Fire models to more sophisticated biophysical models like the HH model.

One of the bases of these models is to consider neurons as circuits. Seen from a circuit's perspective, a neuron's membrane can be seen as a capacitor C_m, separating the ions stored in the neuron from the extra-cellular solution. This separation creates a membrane potential V_m. In a neuron, external currents are allowed to go through the layer via what are called ion channels. Indeed, currents in neurons are not transported by electrons as in typical circuits with conductors but through a variety of different charged ions. A neuron membrane sketch with its ion channels si shown in 2.5. Ion channels open, close, and even can force the transport of ions through the membrane of the neuron depending on different conditions. They do so by sensing the potential difference across the membrane or the concentration of a given chemical in its surroundings and opening, closing, or pumping a certain ion through the membrane. The gating mechanism created by ion channels can be modelled as resistors R_{ion}(or g_{ion} when referring to a conductance) and a voltage source V_{ion} in series. Then, each ion channel can be set in parallel with the capacitor representing the membrane. By using the differential equation for the voltage across a capacitor, the basic equation used in biophysical models can be derived to be $C_m \frac{V_m}{d_t} = I(t)$ where $I(t) = I_{ext} + \sum g_{ion}(V_m - V_{ion})$ is the sum of all currents across the membrane at a given time.

In the following subsections, we will discuss some of the most prevalent models and how they approach the computation of $I(t)$.

2.1.1 Leaky Integrate and Fire (LIF) Model

Leaky Integrate and Fire Models are an attempt at replicating the purely integrative effect of neurons. In a simplified view, a neuron receives signals through time $I(t)$ and integrates this input in the capacitor as the voltage V_m. When the sum of the past received signals is above a threshold V_{th}, a spike is generated and the sum is reset. This is the principle of the Integrate and Fire (IF) neuron. However, neurons do not have infinite memory, receiving small signals separated in time will often not trigger a spike. Likewise, practical capacitors always have a small leakage current that discharges the capacitor. The Leaky Integrate and Fire model (LIF)[6] whose equations can be seen in equation 2.1, replicates this with a leak current that acts as a negative feedback term: $\frac{-(V_m - V_{\text{rest}})}{R}$. The effect of the negative feedback is shown in figure 2.6: the potential decreases between stimulation.

$$\begin{cases} C_m \frac{dV_m}{dt} = \frac{-V_m}{R} + I_{ext}, & \text{if } V < V_{\text{th}} \\ V_m \leftarrow 0,\ \text{Out}_t \leftarrow 1, & \text{if} V \geq V_{\text{th}} \end{cases} \tag{2.1}$$

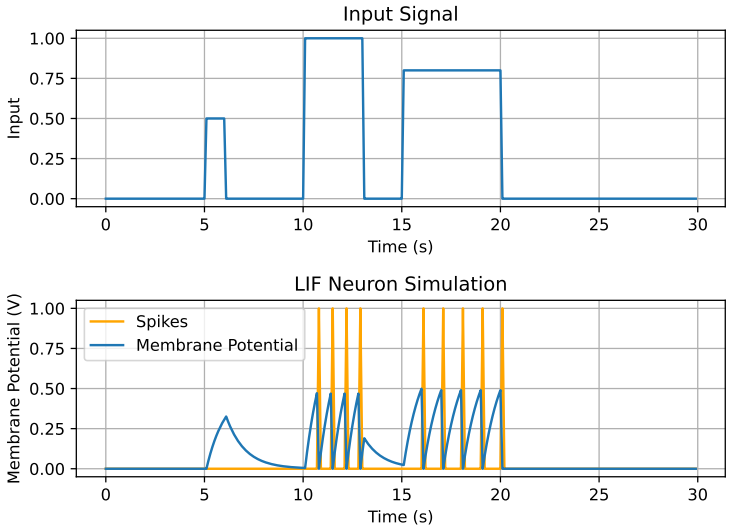


Figure 2.6: Simulation over 300 time steps with a 10ms dt of a LIF neuron with different input intensities and lengths.

LIF neurons have the great advantage of being extremely simple and efficient. This is the main reason for their widespread use in Spiking Neural Networks. However, the sub-threshold dynamics are so simple that they only capture the temporal and spatial input integration aspect that neurons can have. Biological neurons display much richer behaviours that allow them to recognise temporal and spatial patterns in their input. Furthermore, the hard reset used after generating a spike is a pragmatic, non-physiological approximation. In order to accurately model the reset mechanism in biological neurons and all the intricate sub-threshold dynamics and firing patterns, more complex models need to be used.

2.1.2 Conductance Based Models

The Hodgkin and Huxley(HH) model[7] can be seen as the response to the shortcomings of the LIF model. It is the first model based on empirical data. It was obtained by blocking ion channels on the membrane and observing its effects on the input-output behaviour of the neuron. In order to model the impact of the two primary ions causing spikes, Hodgkin and Huxley used a system of 4 differential equations in order to fit the experimental data. The original equations of the model are given in appendix A.1.

The HH model was proposed in 1952 and was based on experimental recordings of the conductances of a squid's neuron. Conductance-based models continue to be developed to refine and expand Hodgkin and Huxley's work by adding ion channels[12] and basing it on other cell types[13]. Introduced in 1977, the Connor Stevens model[13] is a notable example of the expansion of the HH model. It is based on experiments on a crustacean leg axon cell and adds a delayed potassium current I_A. Figure 2.7 shows the circuit corresponding to the CS model with the newly introduced channel. The same model is also expanded in [14] adding calcium currents I_{Ca} to explore their effect on the modulation of the cell. The equations describing the dynamics proposed in this model are shown in equation 2.2. The n, m, h, a, b, m_{Ca} variables model the activation and inactivation of the channels in the gating mechanism. This increased complexity allows the model to simulate 3 different neuron behaviours shown in figure 2.8. The full model and its gating dynamics equations are given in appendix A.3 for conciseness.

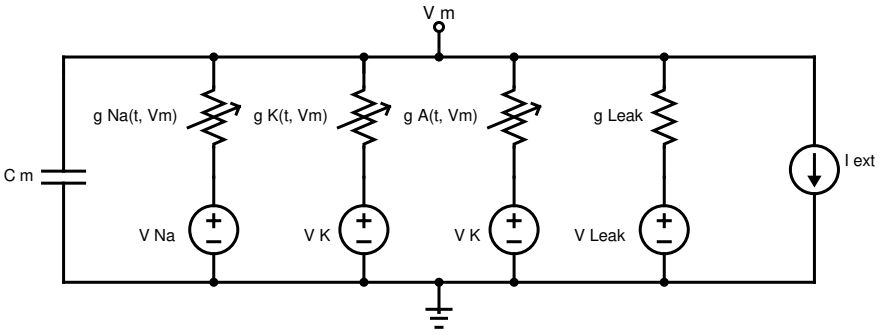


Figure 2.7: Circuit represented by the original Connor Stevens model with Sodium(Na), Potassium(K) and Delayed Potassium(A) currents.

$$
\begin{cases}
C_m \dfrac{dV_m}{dt} = g_l(V_m - V_l) + g_{Na} m^3 h (V_m - V_{Na}) + g_K n^4 (V_m - V_K) + g_A a^3 b (V_m - V_K) + g_{Ca} m_{Ca}^2 (V_m - V_{Ca}) + I_{ext} \\
\dfrac{dm}{dt} = \dfrac{m_\infty(V_m) - m}{\tau_m(V_m)} \\
\dfrac{dh}{dt} = \dfrac{h_\infty(V_m) - h}{\tau_h(V_m)} \\
\dfrac{dn}{dt} = \dfrac{n_\infty(V_m) - n}{\tau_n(V_m)} \\
\dfrac{da}{dt} = \dfrac{a_\infty(V_m) - a}{\tau_a(V_m)} \\
\dfrac{db}{dt} = \dfrac{b_\infty(V_m) - b}{\tau_b(V_m)} \\
\dfrac{dm_{\mathrm{Ca}}}{dt} = \dfrac{m_{\mathrm{Ca}\infty}(V_m) - m\mathrm{Ca}}{\tau_{m_{\mathrm{Ca}}}}
\end{cases}
\tag{2.2}
$$

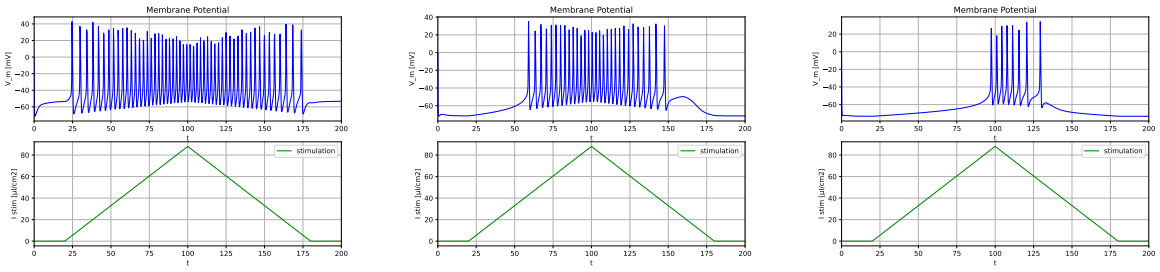


Figure 2.8: Simulation of the Connor Stevens model with 3 different g_A parameter values. From left to right $g_A = 20mS/cm^2$, $g_A = 90mS/cm^2$, and $g_A = 180mS/cm^2$. Using the same ramp input, 3 different behaviours are displayed with 3 different thresholds.

When compared to a model like the LIF, large biophysical models make simulations that are physiologically more accurate. They allow the study of the effects of modulation of parameters on the output with

the advantage of staying in silico. However, the computation of large differential equations systems and auxiliary functions is costly and makes the simulation of large populations unfeasible. To this end, reduced models that retain the main characteristics of neuronal excitability have been, and still are, the subject of considerable research efforts.

2.2 Model Development

The HH model was synthesised by modelling the potassium and sodium currents in a neuron. This was enabled by experiments using the *voltage clamp* technique. This crucial technique can be better understood by seeing a neuron as a dynamically controlled feedback system [15]. From a control perspective, the membrane capacitance and leak resistance act as the plant of the system, while the variable resistors with their voltage source act as the controllers. In this manner, each ionic current contributes to the final feedback signal that results in the generation of spikes. Figure 2.9 shows a sodium and potassium conductance model viewed as a system diagram.

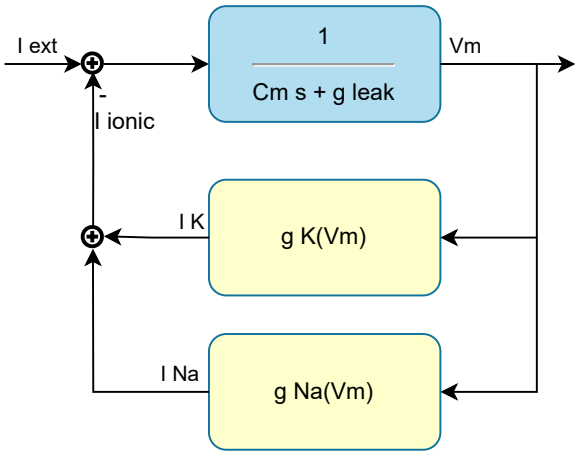


Figure 2.9: Biophysical conductance model with potassium and sodium channels as a feedback system. The system plant is shown in blue and corresponds to the membrane capacitance and leak current. Controllers are shown in yellow and correspond to gating mechanisms that control ionic currents.

2.2.1 Voltage Clamp Experiment

Using the control systems' point of view, we can observe the contribution of each current to the phases generating a spike and classify them by their sign and activation speed. To this end, the voltage clamp experiment was developed as a tool to determine the dynamics of each controller of the system. The principle of the voltage clamp experiment is to observe the step response of the system and obtain a value of $g_{ion}(V,t)$. Figure 2.10 shows the experiment setup. In this set-up, we see that a feedback amplifier is used to maintain ("*Clamp*") the voltage across the membrane V_m to a desired level. The amplifier's output current required to balance the ionic currents through the membrane and keep this voltage is read through an A-meter. Using this setup, the current through a particular voltage-gated channel can be measured by chemically blocking all the other channels. When a step voltage V_{step} is applied, the step response of the specific ionic current $I_{ion}(V_{step},t)$ can be recorded. Then, through the capacitor current equation, we can determine $g_{ion}(V_{step},t)$. By repeating the experiment with different V_{step} values and fitting a curve we determine $g_{ion}(V,t)$, and by repeating the experiment with several channels and aggregating the results we finally obtain biophysical models.

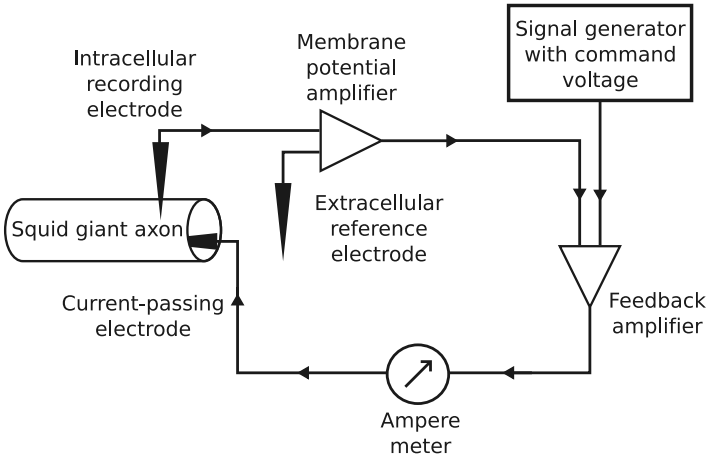


Figure 2.10: Voltage clamp experiment setup. Taken from source

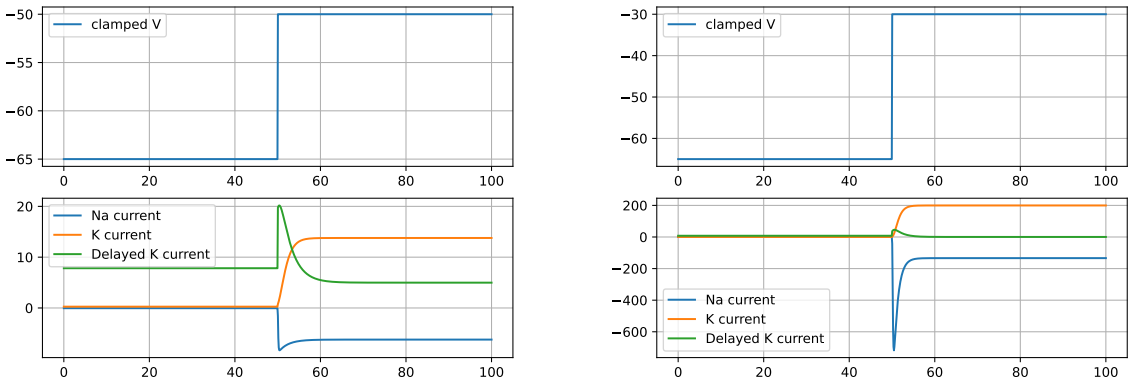


Figure 2.11: Voltage clamp experiment simulation of the Connor-Stevens model with 2 different voltage steps. On the left, the step does not trigger a spike. On the right, a peri-threshold step is given and we can observe the fast increase in Na currents that generate the spike upstroke.

Figure 2.11 shows simulations of the voltage clamp experiment on a neuron modelled using the Connor-Stevens equations. In the voltage clamp experiment, we can see that sodium channels generate a negative current which brings positive feedback that depolarises the neuron to start a spike (regenerative current). Then, potassium channels create positive currents which act as negative feedback that repolarises the membrane (restorative currents). We also see that sodium currents react faster after the step than potassium currents which indicates a slower timescale of potassium compared to sodium. It is this mixed-feedback control across different time scales where the negative feedback takes place after the positive feedback that enables excitability[16]. This pattern can also be observed in the simulation of a spike seen in figure 2.12 where the potassium currents are delayed compared to the sodium currents.

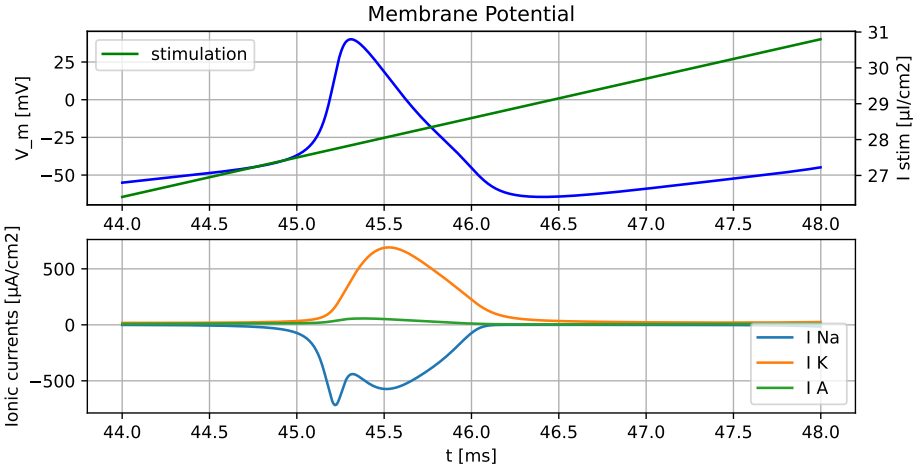


Figure 2.12: Zoom on simulation of a neuron spike using the CS model with $g_A = 20mS/cm^2$. We can see that the potassium currents are delayed compared to the sodium currents

2.2.2 Model Reductions

Voltage clamp experiments based models lead to accurate but large models. As it was previously discussed, this is useful to understand single neurons but it entails drawbacks at the network level due to heavy computational requirements. In order to obtain smaller models that behave like the original large conductance models, 3 main approaches can be taken. The first one is to approximate the conductance model by replacing differential equations with approximate functions. The second one is to aggregate the different ion channels into a few characteristic time scales. Finally, the third one is using qualitative approaches that can be used to develop systems that replicate the excitable behaviour but whose parameters bear no direct relationship with biological neurons.

a Approximation Approach

The main example of model simplification can be seen in [17]. The first approximation that can be done in a conductance-based model is to replace the fastest differential equations with their steady-state equations. In the case of the HH model, $m(t)$ can be replaced by $m_\infty(V)$ with little qualitative differences to the behaviour of the model and removing a differential equation. Furthermore, the authors also make the observation that in the HH model $h(t) \approx n(t) - 0.8$ in a repeated action potential regime. Thus, $\frac{dh}{dt}$ can also be eliminated to reduce the HH model to the 2-dimensional system 2.3 shown here below.

$$\begin{cases} C\dfrac{dV}{dt} = I - \bar{g}_{Na}[m_\infty(V)]^3(0.8-n)(V-V_{Na}) - \bar{g}_K n^4 (V-V_K) - g_L(V-V_L), \\ \dfrac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V)n. \end{cases} \tag{2.3}$$

b Quantitative Approach

Quantitative approaches like [18] and [19] find parameters to aggregate different time scales into a few representative ones. In particular, the Dynamic Input Conductance (DIC) method[19] proposes an algorithm to extract the dynamics of any conductance-based model from voltage clamp experiment data. The algorithm attributes the contribution of each ionic current to up to 3 distinct conductances characterised by their time scales:

- Fast conductance $g_f(V)$: Determined by the fastest depolarisation current time constant usually τ_{Na}.
- Slow conductance$g_s(V)$: Corresponding to the fastest repolarisation current time constant usually τ_K
- Ultra-slow conductance$g_u(V)$: Determined by the slowest time constant of the system.

Each ionic channel X_i can then contribute to the different timescales by their voltage-dependent factor $w_{tscale}^{X_i}(V)$ as shown in equation 2.4

$$\begin{aligned} g_f &= \frac{\partial I_f}{\partial V} = \sum_i w_{fs}^{X_i} \left(\frac{\partial \dot{V}}{\partial X_i} \frac{\partial X_{i,\infty}}{\partial V} \right) \\ g_s &= \frac{\partial I_s}{\partial V} = \sum_i \left(w_{su}^{X_i} - w_{fs}^{X_i} \right) \left(\frac{\partial \dot{V}}{\partial X_i} \frac{\partial X_{i,\infty}}{\partial V} \right), \\ g_u &= \frac{\partial I_u}{\partial V} = \sum_i \left(1 - w_{su}^{X_i} \right) \left(\frac{\partial \dot{V}}{\partial X_i} \frac{\partial X_{i,\infty}}{\partial V} \right), \\ g &= g_f + g_s + g_u = \sum_i \left(\frac{\partial \dot{V}}{\partial X_i} \frac{\partial X_{i,\infty}}{\partial V} \right), \end{aligned} \tag{2.4}$$

This approach not only has the advantage of being general and reducing conductance models to a manageable number of dimensions, but it also quantifies the sensitivity of the behaviour of the system to different ion channels. Intuitively, it gives a measure of how important certain ion channels are in the different stages of generating a spike.

Equations 2.5 show the CS model reduced to only 2 equations using the DIC method [1]. Figure 2.13 shows simulations of the reduced CS model using the DIC method compared to the full model with comparable parameters. As it can be seen, the qualitative behaviour is preserved while the system is reduced to only 2 differential equations. This reduction to 2 dimensions also allows for easier analysis of the dynamics of the system using phase portraits. As we will be seen latter sections, phase portraits are geometrical representations of a system that help visualise and understand its dynamics and where they arise from. Phase portraits usually include the nullclines of a system which are the curves where the equations are at equilibrium. We will see that nullclines and their intersections direct the general behaviour of the system.

$$\begin{cases} C_m \frac{dV_m}{dt} & = I_{ext} - g_L(V_m - V_L) - g_{Na} m_\infty(V_m)^3 h_\infty(m_{Kd,\infty}^{-1}(W)) - g_K w^4 (V_m - V_K) - g_A a_\infty(V_m)^3 b_\infty(m_{Kd,\infty}^{-1}(W)) \\ \tau_n(V_m) \frac{dW}{dt} & = m_{K\infty}(V_m) - W \\ m_{Kd,\infty}^{-1}(V_m) & = -(200 \log(1/V - 21/20))/13 - 208/5 \end{cases} \tag{2.5}$$

[1] It is important to note that the equation $m_{Kd,\infty}$ is not invertible in closed form in the CS model. Therefore, the last equation in 2.5 is the inverse of an approximation using the exponential fit $m_{Kd,\infty}(V) = \frac{1}{1.05 + \exp(-0.065(41.6+V))}$.

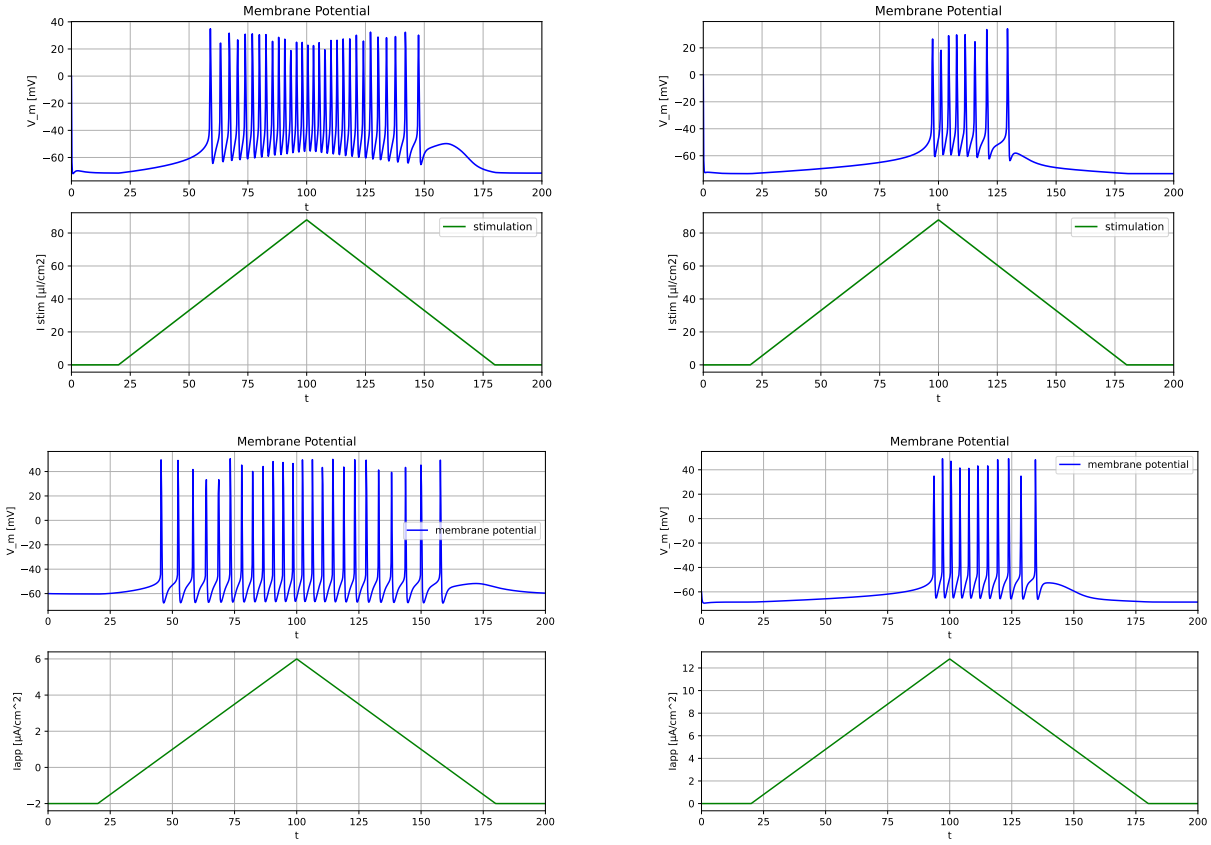


Figure 2.13: Comparison of 2 different spiking regimes of the full and the reduced CS model. Simulations of the full system are on top, reduced model at the bottom. We can see that qualitative behaviour is conserved even though parameter and excitation current values to achieve equivalent behaviour are not identical.

c Qualitative Approach

Seeking a simplified set of equations that would be qualitatively similar to the HH model, the Fitzhug-Nagumo model[20] was proposed. This model was developed using the observations made in the voltage clamp experiment. On one hand, sodium acts in a nearly instantaneous timescale and provides positive feedback when a threshold voltage is crossed. On the other hand, potassium has a slow timescale and provides negative feedback. These assumptions coupled with a geometric approach using phase portraits as shown in 2.14 were used to develop the equations 2.6. Where V_m is the membrane potential, W is a slower recovery variable, and τ is the speed of its change. Using the cubic term V_m^3 term in the fast equation provides a small region of positive feedback surrounded by negative feedback. This creates the contained positive feedback effect. Then, the second equation provides negative feedback that increases proportionally with the difference of V_m and W. The negative feedback also lags behind V_m due to $\tau > 1$.

$$\begin{cases} \dfrac{dV_m}{dt} = V_m - \dfrac{V_m^3}{3} - W + I_{ext} \\ \tau \dfrac{dW}{dt} = (V_m + a - bW) \end{cases} \tag{2.6}$$

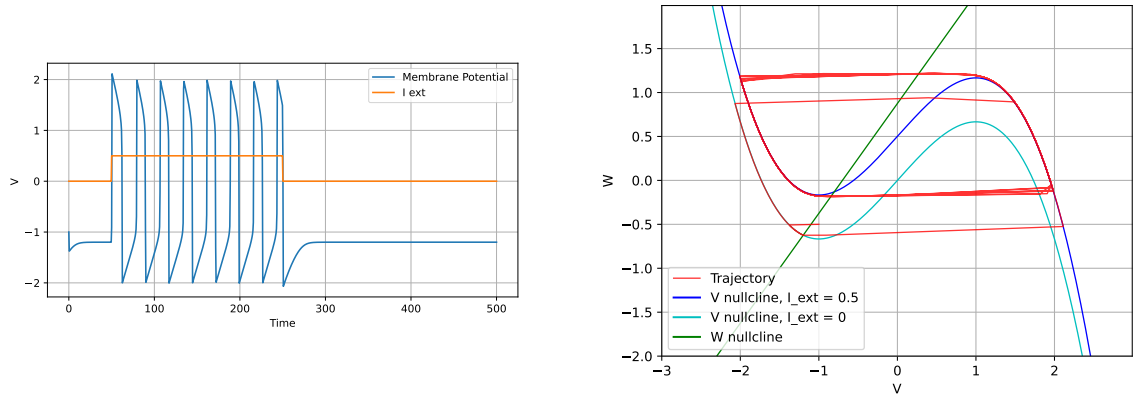


Figure 2.14: Simulation and phase portrait of Fitzhugh-Nagumo model with $I_{ext} = 0.5$

However, the Fithugh-Nagumo model only allows the simulation of one type of excitability. In recent work, [21],[22] and [23] have expanded the equations to model 5 different excitability types. The expanded equations can be seen here bellow in 2.7 where w_∞ is a sigmoid function. In the next section, the 3 main excitability types in the context of this thesis will be further developed from a system dynamics perspective.

$$\begin{cases} \dfrac{dV_m}{dt} = V_m - \dfrac{V_m^3}{3} - (W + W_0)^2 + I_{ext} \\ \tau \dfrac{dW}{dt} = w_\infty (V_m - W) \end{cases} \tag{2.7}$$

2.3 Excitability Types and Modulation

As part of his contributions to computational neuroscience, Hodgekin also proposed the first way of classifying spiking behaviours. He did so by observing the spiking frequency in response to a step input of current [24]. In his experiments, he recognised 3 different firing patterns commonly referred to as types 1, 2, and 3 :

- **Type-1** neurons have a gradual increase in frequency as the input step current is increased. In this sense, their main characteristic is a smooth and continuous *frequency-input current*(F-I) curve.
- **Type-2** neurons are either silent or fire at high frequencies, creating a discontinuity in their FI curve.
- **Type 3** neurons do not repeatedly fire upon the onset of a stimulus. They only fire once, therefore, an FI curve cannot be drawn.

From a signal processing perspective, the FI curves of these neurons show interesting properties. type-1 works as a spiking rate encoder of the input as it provides a continuous relationship between input intensity and frequency. Type-2 provides a step function where the neuron starts rapid spiking if the input is strong enough. Type 3 can be used as specific event start detectors as they will not fire again for sustained inputs.

Figure 2.15 shows the FI curves for 3 increasing values of the delayed potassium conductance g_A in the reduced CS model. From these plots, it would be reasonable to think that the behaviour that is seen is a shift from type-2 to type-1, then to type-2 again with a higher threshold. However, the previously shown current ramp plots in 2.8 correspond to the same g_A values but display different behaviours. Therefore, one can deduce that spiking frequency does not show the whole picture of the dynamics of excitability. In [14] the

behaviour of the rightmost type-2-like simulation is further explored and a new class, type-2*, is introduced to differentiate it from its FI curve analogue type-2 neurons. In figure 2.8, we can see on the ramp-up that the neuron starts firing at a $90A/cm^2$ input. However, it only stops firing after going below $50A/cm^2$ on the down ramp. On the other hand, the leftmost simulation shows a symmetric start and stop to spiking. This is evidence of a mechanistic difference that allows type-2* to have a hysteretic-like behaviour. Hence, this new type introduces memory at a cellular level in neurons which is not present in the two other discussed types. Furthermore, we will see that type-2* also exhibits a delayed response to inputs that allows for the temporal coding of input strength.

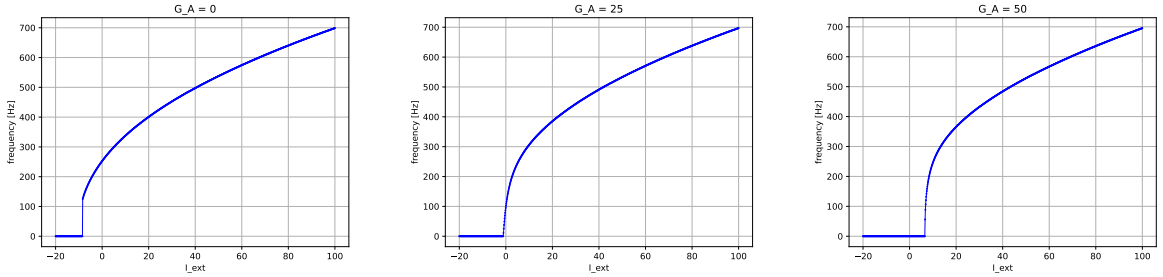


Figure 2.15: FI curves computed on simulations of the reduced Connor-Stevens model for increasing values of g_A.

It is important to note that these behaviours can easily be obtained by tuning a single model parameter in the CS model. This characteristic commonly referred to as neuromodulation is a major contributor to efficient information processing and adequate behaviour in all animals [25]. Indeed, neuromodulation is a key property that models must keep in order to reflect the computational capabilities of biological neurons.

These large qualitative changes in the system when a small perturbation is made to its parameters indicate what is referred to as a *bifurcation*. This type of behaviour is a property of many non-linear systems. Excitability models are a prime example of non-linear systems. It is through the lens of bifurcation theory and phase portraits that the changes taking place to transition between the different excitability types can be explained.

2.3.1 Type-2: Hopf Bifurcation

Type-2 excitability is qualitatively characterised by a discontinuity in the FI curve. This type of excitability is created by a Hopf bifurcation[26] [27] [20]. The bifurcation is caused as I_{ext} increases and the system's fixed point is destabilised, creating a stable limit cycle. This periodic solution is perceived as spiking at a specific frequency. This bifurcation can be seen in the previously discussed simulation of the FHN model in figure 2.14, and also in the reduced CS model simulation in figure 2.16. In both instances, we can see the dip in the V nullcline is vertically translated in the reduced CS model as the stimulation increases. As the fixed point travels right on the nullcline, it changes from stable to unstable after passing the dip. At this point, the dynamics of the system are no longer ruled by the point attractor. Spiking starts as the trajectory opens up and follows a limit cycle that stays close to the V nullcline, switching between the leftmost and rightmost branches.

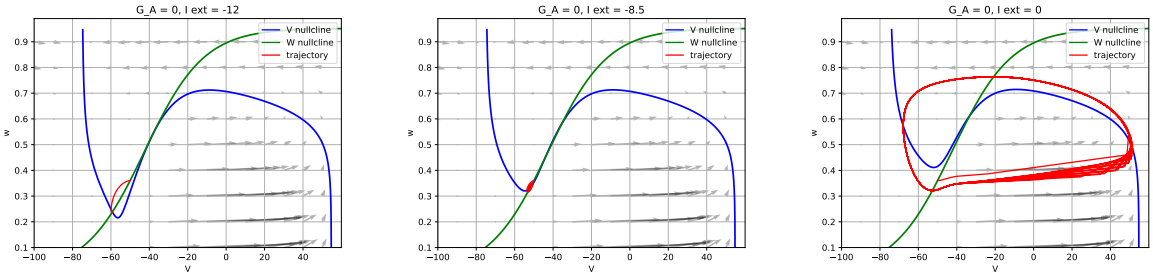


Figure 2.16: Simulation of the reduced Connor-Stevens model using $g_A = 0$ to obtain type-2 behaviour. Stimulation current is increased from left to right to observe the qualitative change of the phase portrait. Currents are respectively below the threshold, at the threshold, and above the threshold. It can be seen that a Hopf bifurcation takes place to initiate spiking.

2.3.2 type-1: Saddle-Node On Invariant Circle and The Organising Centre Of Excitability

type-1 excitability which is qualitatively characterised by a continuous FI curve is created by a Saddle Node on Invariant Circle (SNIC) bifurcation ([28][26][29]). The traditional form of obtaining this bifurcation in neuronal excitability is created by shifting the V-nullcline to create a stable and unstable fixed point that collide annihilating each other when the excitation current is increased. In the case of a SNIC, trajectories that were previously attracted by the stable fixed point now oscillate around a limit cycle that passes through what is referred to as the '*ghost*' of the fixed point. Ghosts can be seen as the remnants of fixed points that still affect the phase portrait even if it they are not present anymore. They do so by decelerating trajectories in its neighbourhood. It is the ghost of the fixed point that slows the limit cycle and allows for low-frequency oscillations close to the bifurcation value of the parameters. Furthermore, it also allows for the continuous modulation of the frequency by going further into the bifurcation, effectively reducing the effect of the ghost stable node. The bifurcation can be seen in the phase portrait of the reduced CS in figure 2.17. However, as it can be seen on the leftmost plot in figure 2.17 the location of the stable fixed point is not on the N-shaped nullcline. A new lower V nullcline branch that intersects the W nullcline emerges as g_a increases. It is at this intersection of both nullclines that the stable fixed point is found. In figure 2.17, as the excitation current is increased we can see that in addition to the previously described SNIC bifurcation, there is also a bottleneck created from the merging of the two V-nullcline branches. This phenomenon further decreases the speed of the trajectory in that region, effectively participating in the qualitative observation of type-1 excitability.

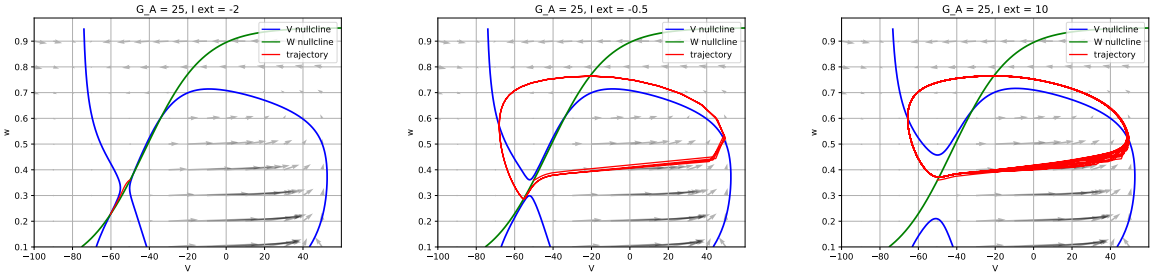


Figure 2.17: Simulation of the reduced Connor-Stevens model using $g_A = 25$ to obtain type-1 behaviour. Stimulation current is increased from left to right. It can be seen that the created limit cycle goes through a bottleneck created by the V nullcline.

The lower branch of the reduction was first observed in [30] using the reduction proposed in [18]. Previous to this observation, type-1 was thought to be obtained solely by shifting the N-shaped nullcline to force the bifurcation to take place along the limit cycle. In [31],[21] and [22], the physiological significance and mathematical properties of the new branch were studied using singularity theory. Its addition was revealed to be fundamental in neuronal excitability by creating a transcritical bifurcation that is at the centre of the 3 discussed excitability types. Not only it is consistent with the mechanisms involved in type-2 and type-1 excitability, but it also creates the ability to reproduce type-2* behaviour that is consistent with recordings of thalamocortical neurons.

From a feedback systems perspective of the CS model, this new branch brings a small positive feedback on the slow timescale. In the case of type-1 excitability, the positive feedback brought by calcium or delayed potassium channels balances the slow negative feedback of potassium channels at the threshold potential. This effectively slows the creation of new spikes in repetitive firing and increases the inter-spike time interval which is the qualitative behaviour observed in type-1.

2.3.3 Type-2*: Saddle-Node Onset, Fold Limit Cycle Termination

In figure 2.18 we can see that increasing g_A conserves the lower branch as in type-1 and further shifts up the phase portrait. Increasing stimulation merges the left and right branches to obtain the previously seen upper N-shaped branch as in type-1. However, unlike type-1 excitability, it does not give rise to a bifurcation . It only shifts an unstable fixed point from the upper branch to the lower branch. Further increasing the stimulation creates a Saddle-Node bifurcation on the lower branch of the V-nullcline that onsets the limit cycle. As it can be seen in the transition from figure 2.18 middle to 2.18 right, the ghost of the stable node and bottleneck effects are reduced as the bifurcation takes place outside the invariant circle which explains the discontinuity in the FI curve. Furthermore, in 2.18 middle it can also be seen that the trajectory follows a stable limit cycle even though the stable FP is still on the lower V-nullcline branch.

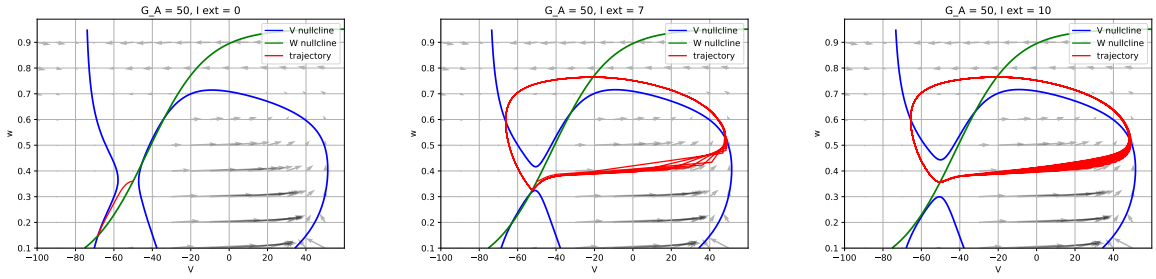


Figure 2.18: Simulation of the reduced Connor-Stevens model using $g_A = 50$ to obtain type-2* behaviour. Stimulation current is increased from left to right. In the middle phase portrait, it can be seen that the stable limit cycle is created while other fixed points are still present in the system. Further increasing stimulation annihilates the 2 other fixed points leaving only the stable limit cycle.

Bistability through the coesxistanc of a stable fixed point and stable limit cycle separated by an unstable fixed point is one of the 2 core properties of type-2* excitability. The second one comes from the onset and termination of spiking at different bifurcations. Termination of the limit cycle occurs when the unstable node creating the separation between the 2 stable regions crosses the limit cycle. This break creates a Hopf bifurcation termination at a stimulation value that is lower than the onset bifurcation value, resulting in hysteresis in the FI curve. Hysteresis can be seen in figure 2.19. From a pure feedback perspective, increasing g_A increases the slow positive feedback experienced by the membrane at voltages close to the threshold potential. Having this small positive feedback promotes the discharge of the neuron after a first

spike is triggered, effectively lowering the threshold after the onset and creating high-frequency spiking and the observed hysteresis in type-2* neurons.

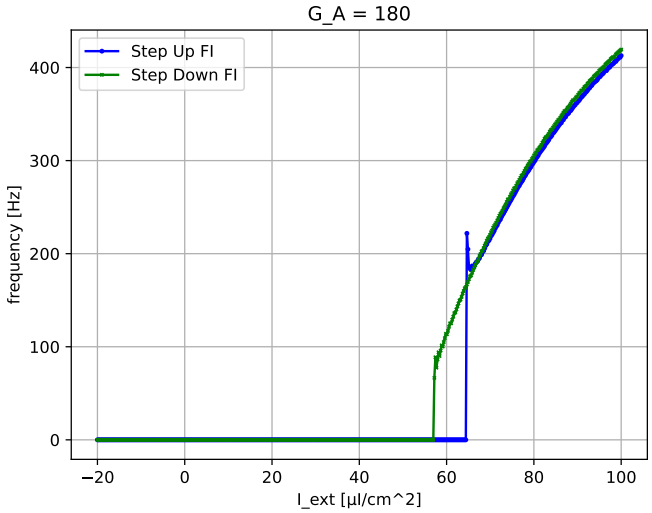


Figure 2.19: Superimposed FI curves of a step up and step down protocol simulated using the full Connor-Stevens model modulated in type-2* excitability. The difference between the two curves is an indication of hysteresis.

2.3.4 Bursting

Bursting refers to a rhythmic pattern of electrical activity exhibited by certain types of neurons. It is characterised by a sequence of spikes, followed by a period of quiescence. The exact roles that bursting takes in the brain is still a current subject of research but it has been linked to processes such as central pattern generation, synchronisation, attention, and sleep to name a few. This firing pattern can be seen in figure 2.20. It can be obtained by adding a third controller to the previously discussed feedback system with a new even slower timescale. To obtain robust bursting dynamics, it has been shown[32] that this timescale needs to be at least 10 times slower than the previously discussed slow timescale. For this reason, the third time scale is referred to as *ultra slow*. The incorporation of the ultra-slow timescale in the previously discussed extended Fitzhugh Nagumo model yields the equations seen in 2.8 where s_∞ and u_∞ are sigmoid functions.

$$\begin{cases} \dfrac{dV_m}{dt} = V_m - \dfrac{V_m^3}{3} - (s+s_0)^2 - k_u u + I_{ext} \\ \tau_s \dfrac{ds}{dt} = s_\infty (V_m - s) \\ \tau_u \dfrac{du}{dt} = u_\infty (V_m - u) \end{cases} \tag{2.8}$$

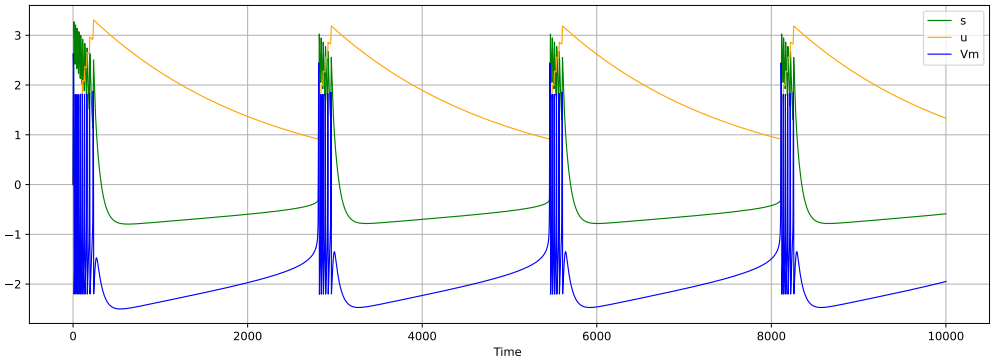


Figure 2.20: Simulation of the extended Fitzhugh-Nagumo model with ultra-slow timescale feedback in bursting regime.

Chapter 3

Machine learning

Machine learning(ML) is a branch of artificial intelligence that focuses on the development of algorithms and models that enable computers to learn from data and make predictions or decisions without explicit programming. It is based on the idea that computers can learn patterns and extract insights from large data sets, allowing them to generalise and perform tasks without being explicitly programmed for each specific scenario. At the core of machine learning are mathematical and statistical techniques that enable computers to learn from data. Machine learning has gained significant attention and popularity due to its ability to tackle complex problems across various domains. The amazing feats that have been achieved are as much a result of improved algorithms than the increased amount of data available and improvements in computing power.

The learning process in machine learning involves training a model using labelled (Supervised ML) or unlabelled (Unsupervised ML) examples, and then using the trained model to make predictions or classify new, unseen data. In supervised learning data is labelled meaning that the desired output or target is known. The model learns to map inputs to outputs by minimising the difference between its predicted outputs and the true labels. Unsupervised learning, on the other hand, aims to uncover hidden insights or clusters in the data without explicit guidance.

Machine learning has revolutionised many industries and has the potential to continue to transform how we solve complex problems. In this chapter, we will give a brief introduction to machine learning techniques focusing on deep learning. Which is a machine learning technique that focuses on learning performed by artificial neural networks (ANNs). We will discuss their evolution through time, and new interesting avenues to solve some of their main limitations.

3.1 Perceptron

In 1943, researchers Warren McCullock and Walter Pitts published a simplified model of a brain cell described as a logic gate, the McCullock-Pitts (MCP) neuron[1]. Based on the MCP neuron, in 1958 Rossenblat. F proposed the perceptron model[33] with an associated learning rule that was able to learn a binary classification of the data points from a labelled data set. This model simplified a neuron to a thresholding unit and synapses as multiplicative factors of the output of incident neurons. Equation 3.1 shows the original function defining a perceptron neuron and figure 3.1 shows a depiction of the working of a single neuron through a computational graph with 3 inputs.

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_i w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

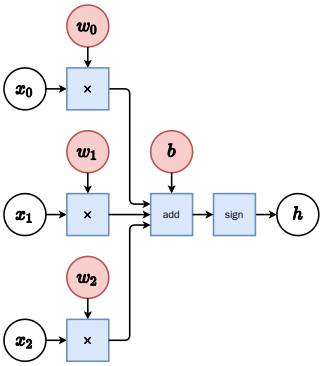


Figure 3.1: Computational graph of a perceptron unit. Taken from Pf.Louppe's INFO8010 course.

Perceptrons can be stacked in parallel to form a layer in order to increase their representation power and perform multiple class classification. However, with a single layer, only linear separation of the data points is possible. Staking layers in series solves this limitation and create a multi-layer perceptron network with the structure that can be seen in figure 3.2. However, the learning rule proposed by Rosenblatt only works for linearly separable problems and one layer of neurons. Solving this issue would require modifications to the base perceptron architecture and the development of more sophisticated learning algorithms.

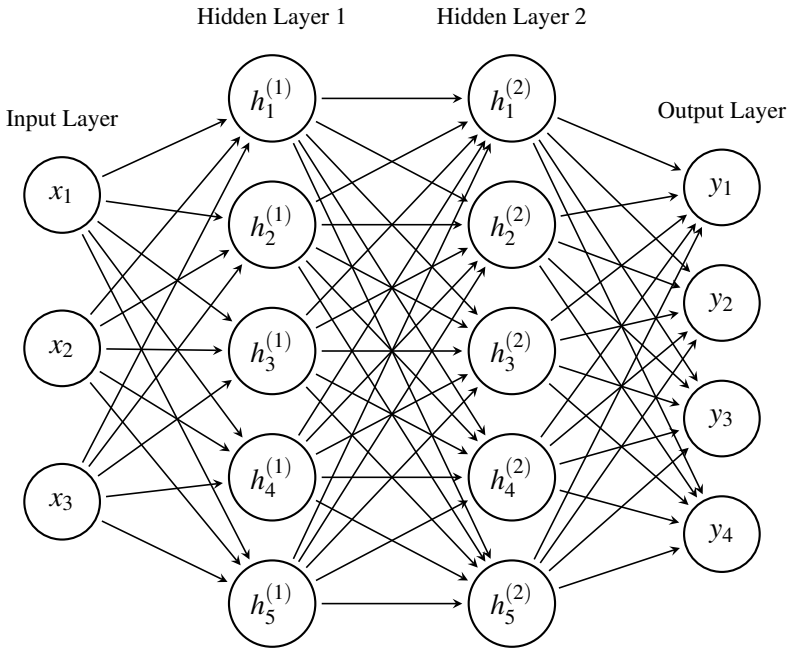


Figure 3.2: Sketch of a fully connected multi-layer perception(MLP) network with 3 input, 2 hidden layers of 5 nodes each and 4 output values.

3.2 Modern Machine Learning

Modern machine learning success can be mainly attributed to two properties of neural networks that make them extremely powerful as learning algorithms. It has been proven that adding layers to a neural network gives them the capacity to be universal function approximations[34], meaning that if there is an in-

put/output relationship in the data, a large enough network can learn it. Furthermore, it has also been shown that increasing the number of layers of a network(increasing its *depth*) improves its maximum achievable performance[35]. However, these two properties could not have been utilised without an algorithm to train deep networks. Two contributions improved the perception concept and learning algorithms allowing deep networks to learn. These improvements have since become the backbone of what we call deep learning.

3.2.1 Introduction of Back Propagation and the Sigmoid Neuron.

In 1960 the ADALINE neural network[36] was proposed where the sign function was removed. Removing the non-linearity of the step function allowed the use of a new learning algorithm, however, it also restricted the network to linear classification as only linear functions of the input could be learned. The new learning algorithm focused on minimising the least squared error of classification given a set of weights w in the network. The learning rule can be seen in the following equation where α is a positive constant that we will call the learning rate, $\hat{y}$ is the network output, x is the input, and y is the ground truth label. This algorithm only worked for a single layer but the concept of minimising the error by tuning the parameters was the precursor of what is used in modern machine learning.

$$w \leftarrow w + \alpha(\hat{y} - y)x \tag{3.2}$$

In order to overcome the linear classification limitation, in the 1970s several researchers replaced the sign function with a sigmoid function as defined here below in equation 3.3 in order to have a smooth and differentiable function that allowed the use of gradient descent.

$$sig(x) = \frac{1}{x + exp(-x)} \tag{3.3}$$

Latter, Hinton et al. [37] generalised and popularised the error minimisation method so that it would work on deep networks through an algorithm called *backpropagation*(BP). The algorithm relies on a function that measures the error of the prediction of the network called a *Loss function*, calculus to attribute the error to the values of the different parameters, and gradient descent to minimise it. This algorithm is at the core of most modern machine learning applications and can be described in 4 steps:

1. **Initialise the network** with a chosen structure and initialise the weights $\mathbf{W}$.

2. **Compute the forward pass**, by feeding an input $\mathbf{x}$ through the *ANN* network to obtain the predicted output $\hat{y} = ANN(\mathbf{W}, \mathbf{x})$. This computation is carried out iteratively layer by layer using the following linear algebra equations:

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$$
$$\mathbf{a}^l = \sigma(\mathbf{z}^l)$$

   Where $\mathbf{W}^l$ are the weights incident to the nodes in layer l, $\mathbf{b}^l$ are the biases, $\mathbf{z}^l$ are the inputs of a layer and $\mathbf{a}^l$ is the output the node with $\sigma(\cdot)$ a nonlinear function. For the input we set $\mathbf{a}^0 = \mathbf{x}$

3. **Compute the loss function** $LF(y, \hat{y})$. The used loss function is selected by the network designer and is problem dependent. The used function is a key element of network design and training that will be further discussed in the following section.

4. **Compute the gradient** $\nabla_{\mathbf{W}} LF(y, ANN(\mathbf{W}, \mathbf{x}))$ of the loss function with respect to the weight in the network. This is done with what we commonly name *the backward pass*. We use the chain rule to iteratively compute the gradient with respect to the weights from the last layer L to the first one.

   At the last layer, we have :

$$\frac{\partial LF}{\partial a^{(L)}}$$

   To obtain the gradient with respect to the weighted inputs z^L using the chain rule with the derivative of the nonlinear function $\sigma'(z^L)$:

$$\frac{\partial LF}{\partial z^{(L)}} = \frac{\partial LF}{\partial a^{(L)}} \cdot \sigma'(z^{(L)})$$

   We can then use iteratively the chain rule to compute the gradient of every layer with :

$$\frac{\partial LF}{\partial z^{(l)}} = \frac{\partial LF}{\partial z^{(l+1)}} \cdot \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}}$$

5. **Update the parameters** in the direction that minimises the loss :

$$\mathbf{W}_{i+1} = \mathbf{W}_i - \alpha \nabla_{\mathbf{W}} LF(y, \hat{y}(\mathbf{W}_i), \mathbf{x})$$

6. **Repeat steps 2 to 5** until the termination condition is met. This could be a finite number of iterations or a convergence criterion on the value of the loss.

3.2.2 Loss functions

Loss functions are a crucial part of the learning process and are problem-dependent. There is a wide variety of loss functions that can be used in machine learning [38], each tailored to specific tasks and objectives. The two broadest problems in supervised learning are regression and classification.

In regression, the goal is to predict a continuous or numerical output value based on input features. The most popular loss function for this kind of task is the previously mentioned **Mean Squared Error (MSE)** loss :

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

MSE is continuously differentiable, making it suitable for optimisation using gradient descent, however, it does penalise large errors heavily due to the squared term, making it sensitive to outliers.

In classification, the goal is to assign input data points to specific categories or classes based on their features. The algorithm essentially learns a decision boundary that separates different classes in the feature space. The output is either discrete and represents the class label or is a probability distribution over classes. The most popular loss function for classification is the **Categorical Cross-Entropy Loss (CCE):**

$$CCE = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

CCE is non-negative, and it converges to zero when the predicted probabilities align with the true class labels. Furthermore, it penalises confident incorrect predictions heavily, encouraging the model to be more certain of its correct predictions.

3.2.3 Activation function

Since the invention of backpropagation, several different nonlinear functions that are applied to the input of a neuron have been proposed [39]. These *activation functions* are one of the essential building blocks of ANNs and have been used to obtain different properties from the neurons. Equation 3.4 shows the general equation describing an ANN neuron where $f(\cdot)$ is the activation function of the neuron.

$$y(\mathbf{x}) = f(\sum_i w_i x_i + b) \tag{3.4}$$

Three of the most commonly used types of neurons in modern deep learning with their properties are the following:

- **Softmax Activation Function** $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$:
  - It normalises the outputs into a probability distribution, where the sum of the probabilities is equal to 1. In the case of 2 classes, it simplifies to the sigmoid function.
  - The softmax function is typically used in the output layer of a neural network for multi-class classification problems.

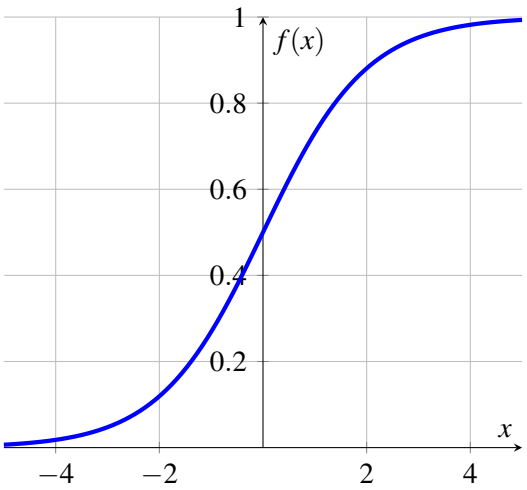


Figure 3.3: Plot of the sigmoid function that is a special case of the softmax function in the case of 2 classes.

- **Hyperbolic Tangent Activation Function** $f(x) = \tanh(x)$:
  - It squashes the input values between -1 and 1, which makes it suitable for outputs that need to be negative or positive.
  - It is zero-centred, which helps with training by having a layer mean output often close to 0.
  - Even though it is less pronounced than in the sigmoid function case, it also suffers from the vanishing gradient problem. The vanishing gradient problem comes from the computation of the gradient when the network is deep and it uses sigmoid-like activation functions. Sigmoid-like functions have derivatives that are bounded in the $[0,1]$ range. Using the chain rule in deep networks multiplies several times this derivative. Effectively, this attenuates the gradient magnitude and impedes learning in the early layers of the network. In figure 3.5 we can see

the derivative of a $tanh(\cdot)$ function multiplied by itself to show that it can become an important source of the attenuation when using the chain rule.

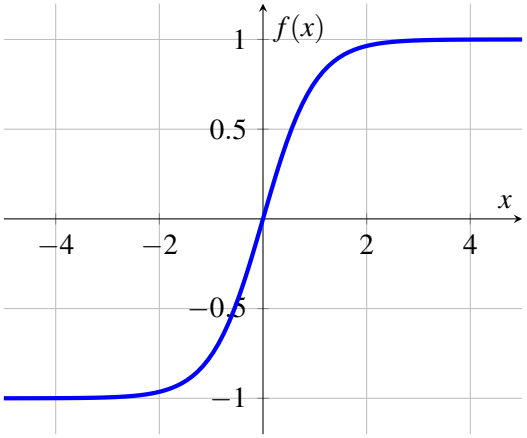


Figure 3.4: Plot of the $tanh(\cdot)$ function.

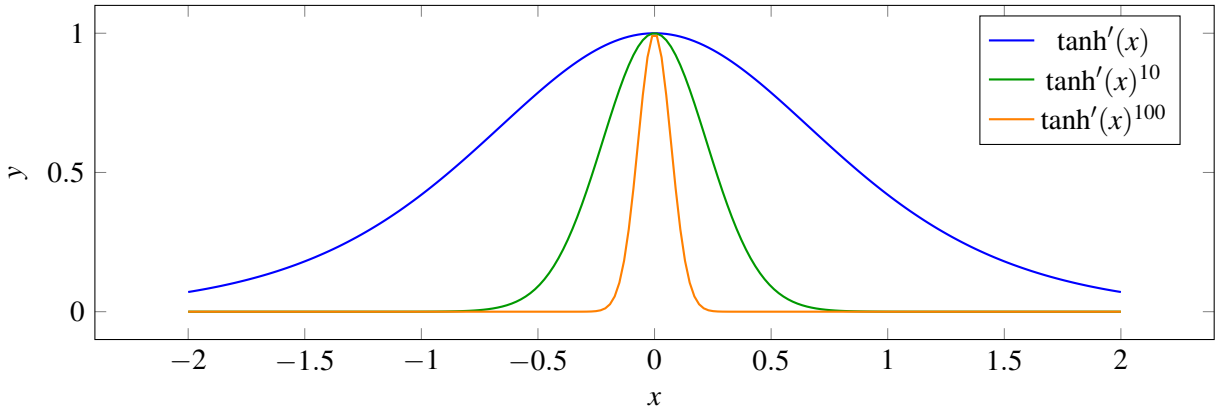


Figure 3.5: Plot of the derivative of the $tanh(\cdot)$ functions multiplied by itself 10 and 100 times. This shows how gradients can vanish and degrade learning in deep ANNs.

- **Rectified Linear Unit (ReLU) Activation Function** $f(x) = \max(0, x)$:
  - It is computationally efficient to compute compared to sigmoid and hyperbolic tangent functions.
  - It does not suffer from the vanishing gradient problem for positive inputs, promoting faster convergence during training.
  - However, ReLU neurons can suffer from the ”dead neuron” problem. when a neuron starts outputting 0, it permanently does so as the derivative with respect to that neuron is also zero in that region and no learning can be induced.

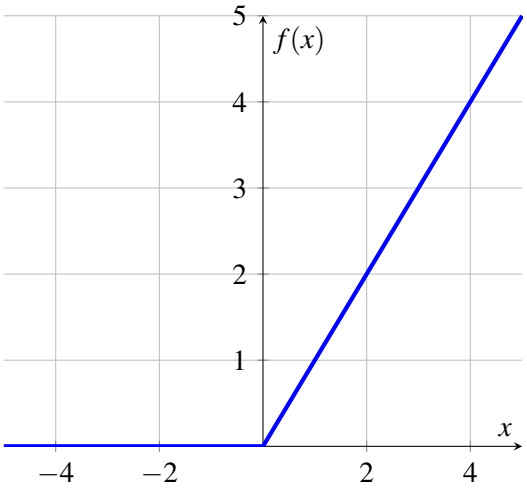


Figure 3.6: Plot of the ReLU function

3.2.4 Parallels Between ANNs and Biological Neural Networks

In the first chapter, we saw the structure of a neuron and the classification of firing types through their FI curve. From the structure, we can see that both have synapses and dendrites that perform spatial integration. However, biological neurons integrate discrete inputs through time whereas ANNs perform matrix-vector multiplications. To bridge this gap, we can see the output of a neuron in an ANN as a frequency value. This parallel is not perfect as negative outputs can be obtained in ANN and there is no such thing as negative frequencies. However, we can see that the non-linearities applied to ANN neurons and FI curves do resemble each other. The FI curve of type 1 neurons and the $ReLU(\cdot)$ activation function essentially both project linearly the input strength. Type 2 neurons on the other hand are thresholding units, akin to the step function in the perceptron of its continuous evolution the sigmoid neuron. Finally, the softmax activation is again a type 2 of neuron with normalisation of neural activity in the layer. However, even in this frequency analogy, we can see that ANNs are not able to recreate temporal integration or other interesting time-dependent properties that biological neurons can have as in type 2*.

3.3 Spiking Neural Networks

The rise of Deep Neural Networks(DNNs) has been able to solve many problems with superhuman performance. However, they have done so by using increasing amounts of data, computing power, and as a consequence huge amounts of energy. Biological brains, on the other hand, use less power due to the sparse nature of communication. It has also been challenging for DNNs to obtain good performance with time-dependent tasks and online learning. These are also two areas of cognition where biological neural networks excel compared to DNNs. To bridge the gap in performance and reduce the energy, data, and compute requirements of modern deep learning, considerable research effort has been directed towards creating machine-learning algorithms that are more closely inspired by biology. With this in mind, a new field of machine learning that uses networks that spike, Spiking Neural Networks(SNNs), has been developed. Nevertheless, this new paradigm creates challenges of its own as the discrete nature of spike times does not allow the direct use of backpropagation to update the weights of the network.

In SNNs, the neuron models seen in the first chapter are simulated and linked together with the goal of performing a specific task. The choice of the neuron model becomes a new design choice for the engineer as more powerful models are able to reproduce more complex signalling but also have many more parameters.

Effectively, this makes them more difficult to simulate and create large networks with. For this reason, most of the SNN literature is based on the LIF neuron model which is one of the most simple and well-understood models. Furthermore, the synaptic connections that transform spikes into currents also have to be modelled and chosen for a given application. The most popular synaptic models in SNNs are defined here below:

1. **Static Synapse Model**:

   This is the simplest synaptic model used in SNNs. It simply transmits the spike modulating its strength with a static weight w.

   The synaptic current generated by a spike at time t_{pre} from the pre-synaptic neuron to the post-synaptic neuron can be represented as follows:

   $$I_{\text{syn}}(t) = w \cdot \delta(t - t_{\text{pre}})$$

   Where w is the fixed weight of the synapse and $\delta(t - t_{\text{pre}})$ is the Dirac delta function, representing the spike event.

2. **Exponential Synapse Model**:

   The exponential synapse model introduces a time decay factor to model the dynamics of synaptic transmission. It models the decay of neurotransmitters as they diffuse in the synapse junction.

   With a spike generated at time t_{pre} from the pre-synaptic neuron, and τ_{syn} the decay time constant:

   $$I_{\text{syn}}(t) = w \cdot \exp\left(\frac{-(t - t_{\text{pre}})}{\tau_{\text{syn}}}\right)$$

3. **Double Exponential Synapse Model**:

   The Double Exponential Synapse Model is an extension of the Exponential Synapse Model, capturing both the rise and decay phases of the synaptic response. It provides a more accurate representation of synaptic transmission dynamics at the cost of increased complexity.

   The synaptic current generated by a spike at time t_{pre} from the pre-synaptic neuron to the post-synaptic neuron can be represented as follows:

   $$I_{\text{syn}}(t) = w \cdot (e^{-\frac{(t - t_{\text{pre}})}{\tau_{\text{rise}}}} - e^{-\frac{(t - t_{\text{pre}})}{\tau_{\text{decay}}}})$$

   where w is the weight of the synapse, τ_{rise} is the rise time constant, τ_{decay} is the decay time constant, and t is the current time.

The effect of some of the most popular synaptic models in SNNs can be seen in figure 3.7.

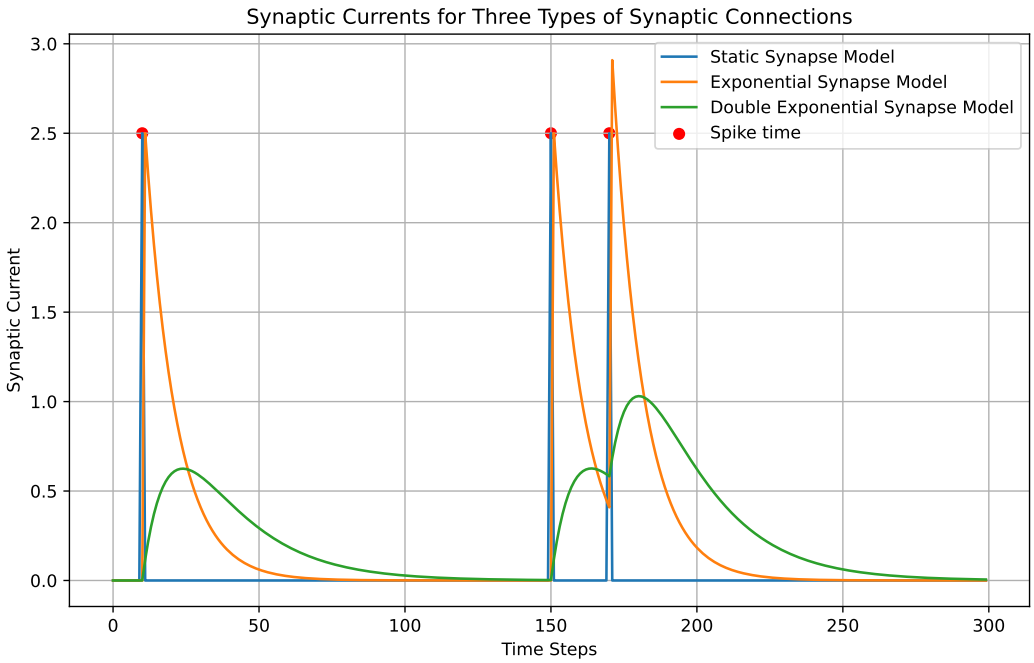


Figure 3.7: Plot of 3 different synaptic models to connect neurons in SNNs.

The afford mentioned synaptic models give a framework that performs time integration of spikes as we observe it in biological neurons. In order to perform spatial integration we simply sum the incident synaptic current to the neuron $I_{in}(t) = \sum_i Isyn_i(t)$

3.3.1 Current SNN Training Rules

As previously mentioned, the largest challenge in spiking neural networks resides in their learning algorithms. The non-differentiable nature of the models renders the use of the well-established backpropagation algorithm unusable in SNNs. To solve this issue, 3 major avenues have been proposed, from the more biologically plausible Hebbian learning rules to the more pragmatic conversions from ANN to SNN techniques.

a Hebbian Learning

Hebbian learning rules are a set of biologically inspired local learning rules based on the principle that "**cells that fire together, wire together.**" They are also commonly referred to as Spike time-dependent plasticity(STDP) rules. They strengthen the connections between neurons that have synchronised spiking activity, thereby enhancing the network's ability to recognise patterns and associations between input signals. The Hebbian learning rule can be stated as follows:

When a pre-synaptic neuron fires and causes the postsynaptic neuron to fire, the synaptic connection between them is strengthened. There is long-term potentiation(LTP) of the connection. Conversely, when the pre-synaptic neuron fires, after the postsynaptic neuron, the synaptic connection is weakened inducing long-term depression(LTD) in the connection. Equation 3.5 shows the formal definition of an STPD rule, and in figure 3.8 the effect of this equation on potentiation can be seen with respect to the spike timings. However, in this definition, the weight can be increased to infinity which is not biologically plausible and

worsens learning by rendering it unstable. In order to limit this effect in STPD rules, homeostatic weighing rules as shown in equation 3.6 are used to preserve the relative weights of the network constant by making the potentiation/depression of connections weight dependent.

$$\Delta w = \begin{cases} A_+ \exp\left(\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_+}\right) & \text{if } t_{\text{pre}} \leq t_{\text{post}} \\ -A_- \exp\left(-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_-}\right) & \text{if } t_{\text{pre}} > t_{\text{post}} \end{cases} \tag{3.5}$$

$$\begin{cases} A_+(w) = \eta_+ \exp\left(w_{\text{init}} - w\right) \\ A_-(w) = \eta_- \exp\left(w - w_{\text{init}}\right) \end{cases} \tag{3.6}$$

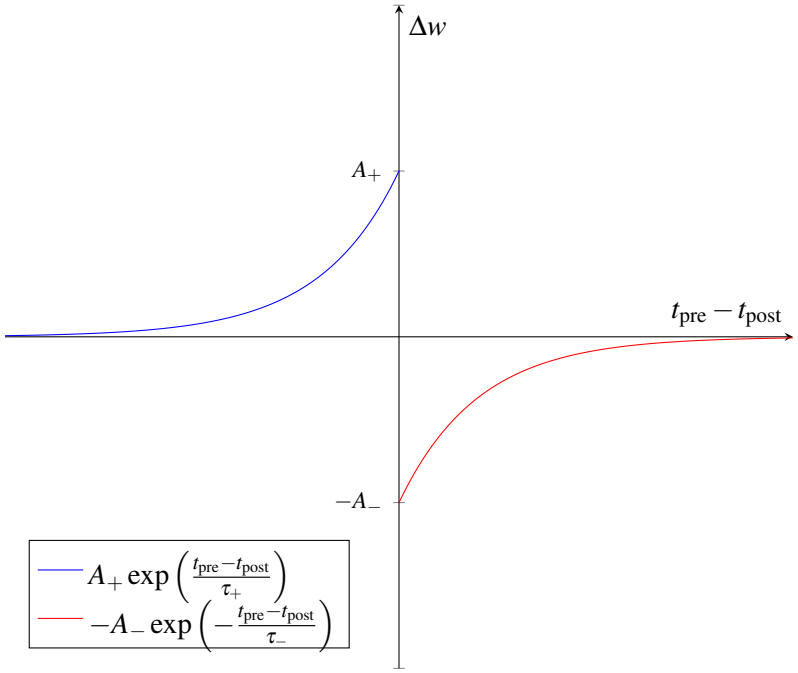


Figure 3.8: Plot of a Spike Time-Dependent Plasticity Rule showing how the pre-post synaptic spike time interval affects the potentiation of the connection Δw.

Many variations of STDP rules have been proposed with varying success. The unsupervised aspect of this technique with its asynchronous nature makes it very appealing as a biologically plausible learning rule. Nevertheless, the locality of this technique makes it challenging to obtain state-of-the-art performance for tasks that require larger networks. Furthermore, there is no learning if there is no spike which can lead to problems similar to the ReLU dead neuron problem.

b Surrogate Gradient

Bio-plausibility of backpropagation and how it would manifest in the brain is a controversial topic in neuroscience. However, it has been proven to work well and contrary to STDP, it is a global error signalling that is able to attribute blame specifically the weights that will minimise the error of the task at hand.

The surrogate gradient descent method originally proposed in [40] makes the use of gradient descent possible in SNNs by replacing the non-differentiable thresholding function that neurons use to trigger spikes. It does so only during the backward pass of the backpropagation algorithm by replacing it with an approximate continuous function. This approximation allows the use of backpropagation but adds one

more complexity to the design of an SNN, which is the choice of the surrogate function used. Furthermore, this technique is an approximation which means that it is sensitive to the chosen surrogate function and that there is a possible loss of performance due to it. Another disadvantage of the smoothing of the threshold function comes from the participation of subthreshold dynamics in the weight update of non-firing neurons. This can further decrease the learning performance of the SNN. Even though surrogate gradient descent obtains better performances than using STDP rules, ANNs still outperform SNNs that use this technique.

c ANN to SNN Transfer

Another solution to train SNNs with backpropagation is to use the analogy previously made of how ANN neurons are similar to biological neurons. An ANN can be trained using a backpropagation and then the weights can be swapped into an equivalent SNN as originally proposed in [41]. This conversion is usually done between ReLU units in the ANN and IF neurons in the SNN.

ANN to SNN conversions are a convenient way to train networks using known techniques and then transform the network to be more energy efficient at inference time if used on neuromorphic hardware. However, there are several drawbacks to training an SNN in this way. First of all, the transfer makes neurons that fire often as they only encode knowledge in the firing frequency and do not use spike timings. Not only this is less energy efficient but it also cripples the achievable performance of SNNs. Indeed, doing ANN to SNN transfer sets a maximal performance limit as the SNN will not be able to perform better than the ANN it is converted from.

3.3.2 Input Encoding

SNNs face a significant challenge due to the nature of the data they operate on. Unlike ANNs, where data is commonly in the form of digital values, SNNs require input data in the form of spikes. However, the availability of neuromorphic sensors that directly produce spikes is still limited, making it necessary to transform digital data sets commonly used for ANNs into a format suitable for SNNs. This transformation process is crucial, as the disparity in performance between ANNs and SNNs could, in part, be attributed to the sub-optimal representation of input data in SNNs.

To bridge this gap and enable proper data handling in SNNs, two commonly used approaches for converting digital data to spiking data are rate coding and latency coding. The effect of the different encoding styles on 5 values can be seen in figure 3.9.

a Rate Coding

Rate coding entails representing information in SNNs based on the frequency or rate of spikes. It translates continuous values from the digital data set into spike rates, where higher rates indicate higher intensity. Rate coding is a simple and easy to implement manner to encode digital data. However, the precise timing of individual spikes is not considered, potentially affecting the overall performance of the SNN. Furthermore, this usually leads to a very dense spike encoding, reducing the energy efficiency benefits of SNNs.

b Latency Coding

Unlike rate coding, latency coding considers the precise timing of spikes as crucial information. In this approach, the relative timing of spikes encodes the data. It can provide better temporal precision and potentially higher efficiency in information representation for SNNs than with rate coding, leading to improved performance.

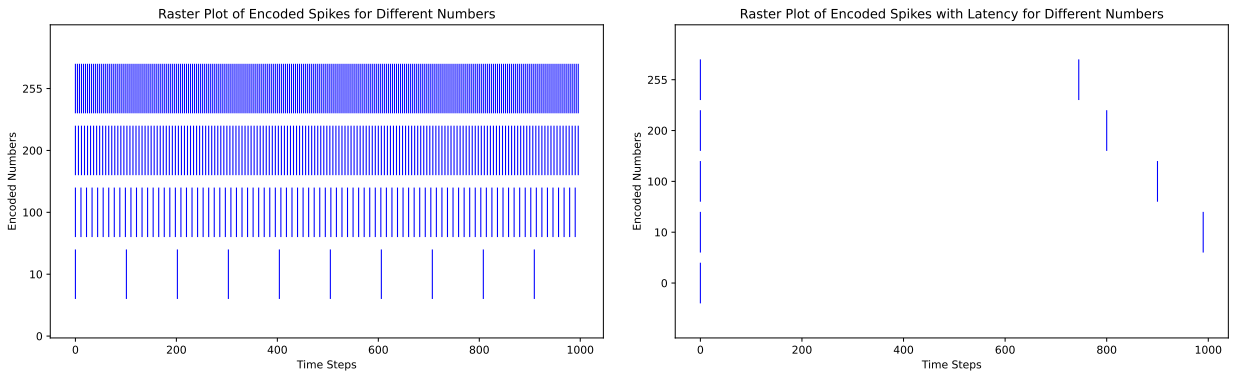


Figure 3.9: Comparison between the coding of 5 different values using rate coding on the left vs latency coding on the right over 1000 time-steps.

3.4 Benchmarks

As the complexity and variety of machine learning models grow, it becomes increasingly important to not only assess their performance but to easily compare their effectiveness in a standardised manner with other techniques. This is why benchmarks were created in the field. Benchmarks are standardised data sets, tasks, or evaluation protocols used to measure and compare the performance of different models. They serve as a fair common ground for researchers to understand the strengths and weaknesses of their models, allowing them to identify areas for improvement. Moreover, benchmarks help in establishing a baseline performance for specific tasks, making it easier to gauge the progress made in the machine learning community over time. Some of the most popular benchmarks like MNIST[42], CIFAR[43], or IMAGENET[44] assess image classification but there are many more that asses tasks such as machine translation and image segmentation to name a few.

3.4.1 MNIST

In particular, the MNIST (Modified National Institute of Standards and Technology) database was created in the late 1990s but has become a cornerstone in the development and evaluation of image classification algorithms. The dataset consists of handwritten digits (0 to 9) resized to 28x28 pixels grayscale images as can be seen in figure 3.10. With 70,000 images, MNIST is small compared to modern datasets, yet, its simplicity and small size allow researchers to use it as a fast first assessment for their algorithms.

3.4.2 Performance SNNs Learning Rules on MNIST

The MNIST benchmark can be used to show the performance obtained with some implementations of the previously discussed SNN learning algorithms. Table 3.1 summarises some of the performance of published SNNs and their training approach on MNIST adapted from the tables in [45]. As we can see, ANN to SNN transfer is what works best, followed by surrogate gradients, and STDP-based rules. It is also important to mention that the architecture of the networks has a significant impact on their performance. Indeed, we can see that convolutional networks generally work better on image recognition tasks than fully connected ones.

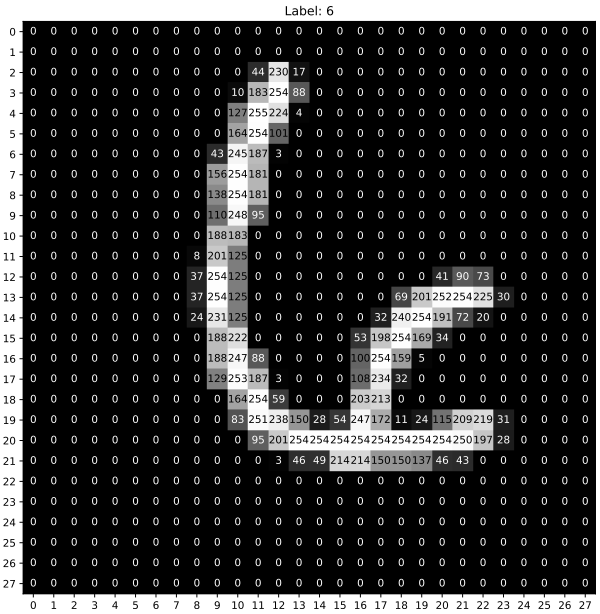


Figure 3.10: MNIST database sampled image with displayed pixel value and label.

Table 3.1: Summary of SNN network performance on the MNIST dataset and their training method adapted from [45].

Network	**Method**	**Accuracy (%)**	**Reference**
Surrogate gradient Fully connected networks			
FCN (169–500–10)	MT-10 (heu/noheu)	99.10	[46]
FCN (784–800–800–10)	BP with rate-coded	98.66	[47]
FCN (784–800–10)	SLTCSNN	97.20	[48]
Surrogate gradient Convolutional networks			
LeNet	Spike-based BP	99.59	[49]
CNN (32C3-32C3-64C3-P2-64C3-P2–512-10)	IIR-SNN	99.46	[50]
CNN (5,32,2-5,16,2–10)	SNN+DT	99.33	[51]
STDP based learning on fully connected SNN			
FCN (784–4500–10)	VPSNN STDP	98.52	[52]
2-Layer (784–6400)	Exponential STDP	95.00	[53]
MLP (784–1600-10)	SIF-STDP (Q2PS)	89.70	[54]
STDP based learning on convolutional SNN			
ConvNets (2Cov.+3FCN)	STDP+BP	98.60	[55]
Spiking CNN (H128-D32)	Probalistic-STDP	98.36	[56]
CSNN+SVM	THRADA-STDP	98.60	[57]
ANN to SNN transfer FCN Networks			
FCN	Data-Based	98.64	[58]
FCN	Model-Based	98.61	[58]
ANN to SNN transfer CNN Networks			
SNN	SNN	99.44	[59]
VGG-19	M-SNN	99.57	[60]
CNN	CNN(LRN/noise)	99.09	[61]

Chapter 4

Recurrent Neural Networks

Many real-world problems entail data of a sequential nature. Some examples are dialogues, stock market valuations, or even neuron potential. With the goal of processing sequential data, in the 1980s Recurrent Neural Networks (RNNs) were proposed as a solution to the problem. RNNs are neural networks whose output is (partially)fed back to the input and inputs are fed in a sequential fashion. In this way, RNNs are able to handle inputs of varying lengths and create a form of memory of past inputs. They are also able to establish dependencies with respect to the context and order in which data is fed to the network. In these networks, the output can be extracted after each token is fed in order to later make a prediction. This new neural network architecture was used for different tasks such as sequence classification, sequence synthesis, or sequence-to-sequence translation. These tasks have applications like, text sentiment analysis, music synthesis, and stock price forecasting.

Recurrent nets were first proposed and then formalised by [62] in 1969 and [63] in 1988 respectively. Since then, much research has developed the idea and many different recurrent networks have been proposed to solve different shortcomings in RNNs. In this chapter, we will go over the principles behind RNNs, the challenges that this architecture face, and some of the principal contributions related to the mSRC, the developed cell in this thesis.

4.1 RNN Working Principles and Challenges

The base structure of an RNN network can be seen in figure 4.1. Any feed-forward ANN can become an RNN if a feedback connection is made by taking its output and feeding it to the input. The simplest architecture for an RNN would be a multi-layer perception where the outputs of the network are fed back into the inputs with the next input token. Nevertheless, vanilla RNN architectures like these are known to be difficult to train. They can suffer from vanishing gradients, exploding gradients, and often have issues capturing long-term dependencies in sequences.

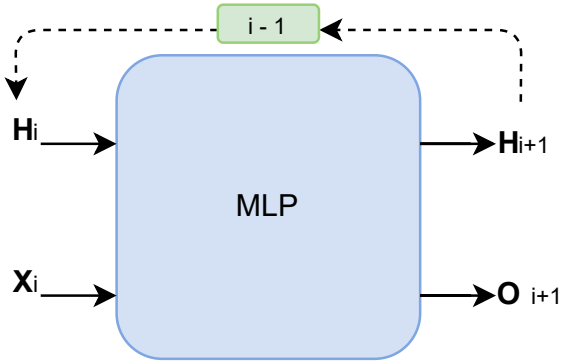


Figure 4.1: Structure of a simple recurrent neural network composed of an MLP. At every element of the sequence, it takes an input vector X_i and a hidden state vector H_i, and outputs the next hidden state vector H_{i+1} and an output vector O_{i+1}.

4.1.1 Backpropagation Through Time (BPTT)

Back-Propagation Through Time (BPTT) is an extension of the traditional back-propagation algorithm. In BPTT, the RNN is unfolded over time, creating a chain of interconnected neural networks, one for each time step. The objective is to compute the gradients of the loss function with respect to the parameters so that they can be updated through gradient descent even though the network has a feedback connection.

The unfolded RNN can be depicted as seen in figure 4.2. It is important to note that for long sequences, the unfolded network can therefore be very deep, which creates challenges during training.

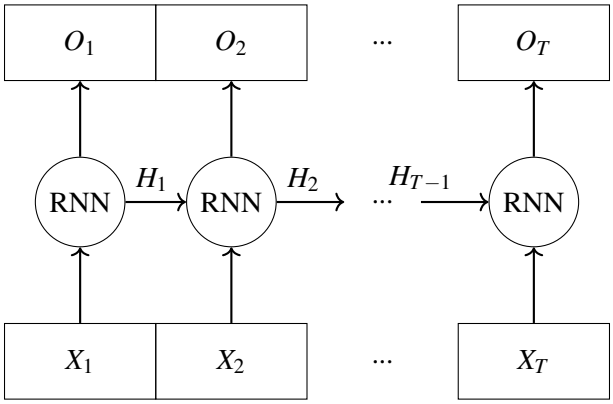


Figure 4.2: Depiction of an unfolded RNN in order to perform backpropagation through time.

4.1.2 Vanishing Gradients

Vanishing gradients occur when the gradients become very small as they are propagated back through time during BPTT. This phenomenon is particularly problematic in vanilla RNNs and is mainly attributed to the nature of the activation function used in the network, typically the $\tanh(\cdot)$ function.

In RNNs, the vanishing gradient problem becomes especially severe when processing long sequences, as the gradients may become virtually zero after propagating through numerous time steps, preventing the network from learning long-range dependencies.

4.1.3 Exploding Gradients

Conversely, exploding gradients occur when the gradients become extremely large during BPTT. This phenomenon often happens when the weights in the network are too large, causing the gradients to explode exponentially as they are propagated back through time.

When the gradients become too large, the optimisation process becomes unstable, and the model's parameters can diverge, leading to training failure.

4.2 How to Attenuate the Challenges Observed in RNNs

In order to attenuate the afford mentioned challenges, researchers have proposed various solutions, including Gated RNN architectures like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) that reduce the effect of vanishing gradients, gradient clipping that will scale down large gradients, and normalised initialisation of weights that also attenuates the gradient problems.

4.2.1 Gated RNN Architectures

Gated RNN architectures, such as LSTM and GRU, are designed to address the vanishing gradient problem and attempt to capture long-term dependencies more effectively. Their architectures can be seen in figure 4.3 and 4.4. Their main property is that they have what is commonly referred to as a "gradient highway" which is a path for the gradient to go through the cell without an activation function that will attenuate it.

a LSTM (Long Short-Term Memory)

The LSTM architecture was introduced by Hochreiter and Schmidhuber in 1997 [64]. It is a memory cell that introduces an internal state C_t whose function is to retain information over time, and a gating mechanism that will selectively modify this state. The internal state of the cell is modified according to the output of 3 small NN layers that we call gates, namely, the input gate, forget gate, and output gate.

The equations describing the LSTM cell are given equation 4.1 where x_t is the input at time step t, h_t is the hidden state, W and b denote weight matrices and bias vectors respectively, σ and tanh are the layers' activation function, and $\odot$ is element-wise multiplication.

$$\begin{cases} i_t = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i) \\ f_t = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f) \\ o_t = \sigma(W_o \cdot [x_t, h_{t-1}] + b_o) \\ \tilde{C}_t = \tanh(W_c \cdot [x_t, h_{t-1}] + b_c) \\ C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ h_t = o_t \odot \tanh(C_t) \end{cases} \tag{4.1}$$

In the LSTM equations, we can see that the internal state C_t does not go through a squashing function to attenuate the vanishing gradient problem and improve the formation of long-term dependencies during training. Furthermore, we can see that there are 4 different sets of weights and biases that are involved in the gating mechanism. Namely: W_i, W_f, W_o, and W_c. These high amounts of complexity contribute to the LSTMs' performance but also slow training as for each token of the sequence, the output has to be computed during the forward pass and then its gradient has to be computed during the backwards pass.

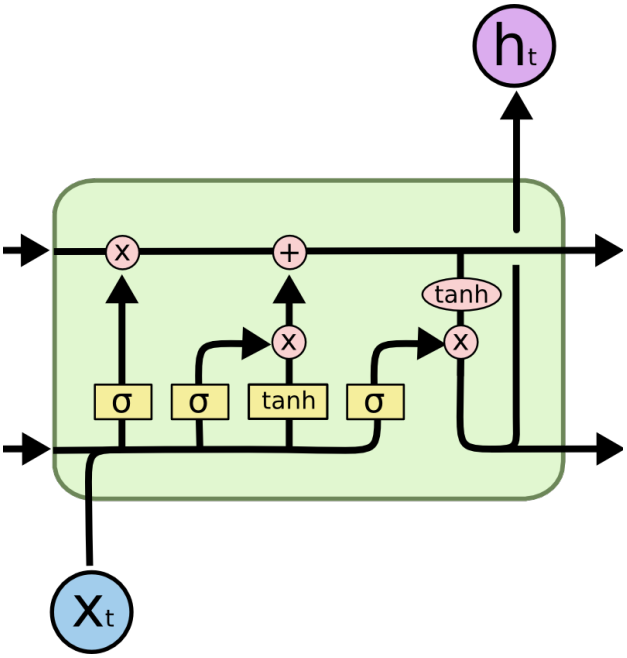


Figure 4.3: Sketch on the architecture of an LSTM cell obtained from source

b GRU (Gated Recurrent Unit)

The Gated Recurrent Unit (GRU) was introduced by Cho et al. in 2014 [65] as a simplified version of LSTM that still has a gradient highway. It uses two gates: the reset gate r_t and the update gate z_t. Due to the reduced amount of gates involved, it is computationally less expensive than LSTM while still achieving competitive performance.

The GRU cell equations can be seen in equation 4.2 here below, and a sketch of the cell is shown in figure 4.4:

$$\begin{cases} r_t = \sigma(W_r \cdot [x_t, h_{t-1}] + b_r) \\ z_t = \sigma(W_z \cdot [x_t, h_{t-1}] + b_z) \\ \tilde{h}_t = \tanh(W_h \cdot [x_t, r_t \odot h_{t-1}] + b_h) \\ h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{cases} \tag{4.2}$$

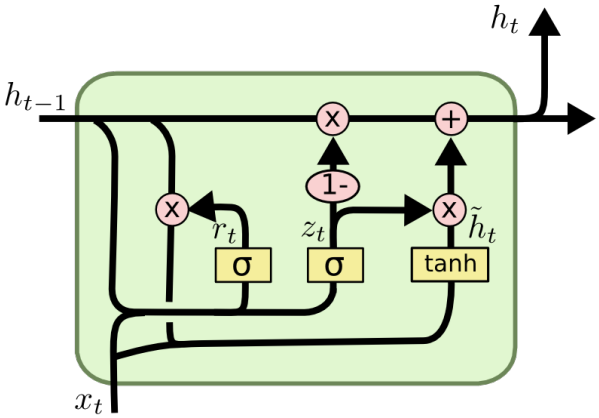


Figure 4.4: Sketch on the architecture of a GRU cell obtained from source.

4.2.2 Gradient Clipping

Gradient clipping is a technique used to prevent exploding gradients during training. It involves measuring the gradient and scaling it down if its norm exceeds a predetermined threshold th. This ensures that the gradients remain within a reasonable range, preventing the model's parameters from diverging during training.

Let $\mathbf{g}$ be the computed gradient vector, $||\mathbf{g}||$ be its Euclidean norm, and $\hat{\mathbf{g}}$ the value of the gradient used for the update. The gradient clipping process is described by equation 4.3:

$$\hat{\mathbf{g}} = \begin{cases} \mathbf{g} & ,||\mathbf{g}|| \leq th \\ \frac{\mathbf{g}}{||\mathbf{g}||} * th & ,otherwise \end{cases} \tag{4.3}$$

4.2.3 Orthogonal Initialisation of Weights

Orthogonal initialisation is another technique used to mitigate both the vanishing and exploding gradients problem. It initialises the weight matrices in such a way that they have orthogonal columns. Orthogonal matrices have the property that their singular values are all equal to 1, which helps in preventing the gradients from vanishing or exploding during backpropagation.

This can easily be proven in the case of a simplified RNN with identity activation functions, 0 input and 0 bias. In that case, we can define the hidden state h_t as :

$$\begin{aligned} h_t &= W_x \cdot x + W_h \cdot h_{t-1} + b \\ \iff h_t &= W_h \cdot h_{t-1} \end{aligned}$$

For a sequence of length n and considering an identity initial hidden state $h_0 = I$, we have :

$$\begin{aligned} h_n &= W_h \cdot (...(W_h \cdot (W_h \cdot h_0))...) \\ \iff h_n &= W_h^n \cdot h_0 \\ \iff h_n &= W_h^n \cdot I \\ \iff h_n &= W_h^n \end{aligned}$$

With an orthogonal initialisation, the matrix is diagonalisable to $W_h = S \cdot \Lambda \cdot S^{-1}$ where Λ is a diagonal matrix of the eigenvalues of W_h and it has unit (1 or -1) eigenvalues. Therefore, $h_n = W_h^n = S\Lambda S^{-1}$ remains bounded and attenuates the vanishing and exploding gradients problem.

4.3 Feedback Nature of RNNs and How to Harness the Possibilities it Offers

Even though the feedback connection that characterises RNNs makes them feedback systems, little work has gone into their understanding under this light. In [66] RNNs are trained to replicate systems with known behaviours and classical techniques in nonlinear dynamics analysis such as dimensionality reduction, fixed point analysis, and phase portraits are used to characterise the dynamics of the RNN. In this paper, they showed that RNNs do create fixed points and saddle points to generate memory and transition between states in a similar manner to the systems they reproduced.

4.3.1 Brain-inspired RNN cells

In [66], they trained known systems into classic RNN architectures to replicate them. However, it was also shown that the learnt systems did not reproduce identically the behaviour of the original systems leading to degraded performance. This opened the question of whether RNN cells can be designed following feedback system dynamics and add learned elements to these systems in order to perform a desired task harnessing the properties of the dynamical system.

a Bistable Recurrent Cell (BRC) and network BRC(nBRC)

In a first attempt to answer the previous question, the Bi-stable Recurrent Cell(BRC)[67] was designed. The BRC was thought with the intent to harness bi-stability, a nonlinear system property that allows for never-fading memory that is notably found in neurons[68]. Never-fading memory is a highly desired property in recurrent neural networks as one of their main challenges is learning long-term dependencies. The equations defining the BRC cell can be seen in equations 4.4 here below, and a sketch of the cell is also shown in figure 4.4.

$$\begin{cases} a_t = 1 + \tanh\left(U_a x_t + W_a h_{t-1}\right) \\ c_t = \sigma\left(U_c x_t + W_c h_{t-1}\right) \\ \hat{h}_t = \tanh\left(U_h x_t + a_t \odot h_{t-1}\right) \\ h_t = \left(1 - c_t\right) \odot h_{t-1} + c_t \odot \hat{h}_t \end{cases} \tag{4.4}$$

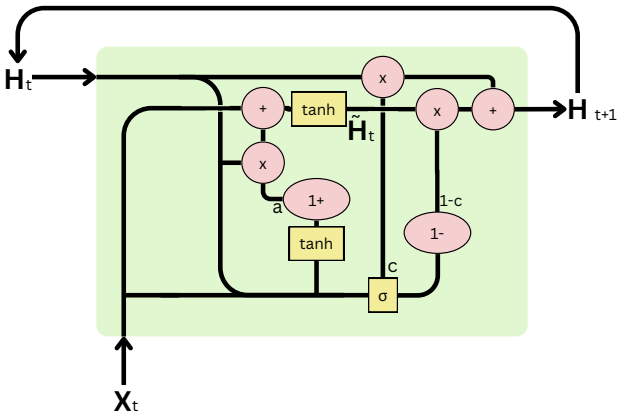


Figure 4.5: Sketch of the BRC cell architecture.

We can see that the BRC constrains the GRU to either obtain a faithful or categorical representation of the input and creates memory through a pitchfork bifurcation. The update equations of the nBRC in 4.4 show a as the bifurcation parameter that is contained in the range $[0, 2]$. For values below 1, a acts as negative feedback, therefore, making the system produce a faithful representation of the input. For values above 1, the update will exhibit positive feedback creating a categorical representation of the input by pushing the state of the cell to either 1 or -1 depending on the input value. The range [0,2] is chosen so there is an equally sized range of values for faithful and categorical representation of the input. Figure 4.6 shows the effect of the bifurcation parameter on a sinusoidal input. For values of a close to 2, the memory is difficult to change creating that never fading memory.

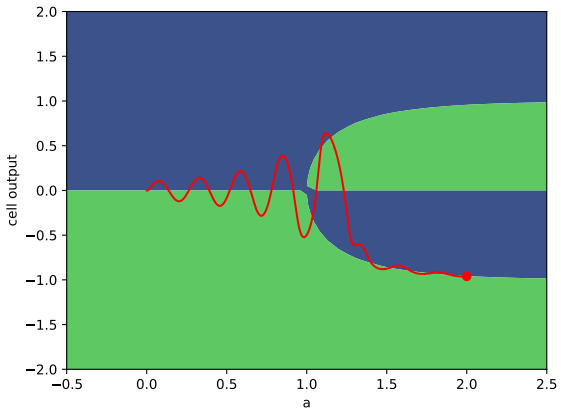


Figure 4.6: Bifurcation diagram showing the pitchfork bifurcation present in the nBRC equations. The trajectory in red is generated through 500 time-steps with a sinusoidal input and linearly increasing value of a. It can be seen that the representation is faithful for values below 1 and becomes categorical as a increases.

In practice, the cell showed satisfactory results in the generation of never-fading memory matching and outperforming GRU and LSTM cells in certain long-term memory tasks. More importantly, it was the first confirmation that known nonlinear dynamics could be used to design RNN cells and that their properties could be harnessed through learning with backpropagation. These findings paved the way to develop more bio-inspired RNN cells.

b Spiking Recurrent Cell (SRC)

Recently, the idea of creating a bio-inspired RNN cell was expanded to create a cell that would perform spikes like biological neurons do. In [69] the Spiking Recurrent Cell(SRC) adds a negative feedback to the BRC that allows it to perform spikes. Indeed, when the BRC is set to be bi-stable with $a = 2$, the cell has local positive feedback. As it has been seen in the chapter about excitability, this is the main mechanism in a biological spike upstroke that is led by sodium channels. Adding slower, negative feedback to the system will then bring the cell to its rest state as the down-stroke does in a biological neuron with potassium channels. Furthermore, they propose a synapse connection that recreates the previously discussed exponential synapse model to integrate the spikes at the input of the neurons. Equation 4.5 and 4.6 show the cell and synapse equations respectively.

$$\begin{cases} h[t] & = z \odot h[t-1] + (1-z) \odot \tanh\left(x[t] + r \odot h[t-1] + r_s \odot h_s[t-1] + b_h\right) \\ z_s[t] & = z_s^{\text{hyp}} - \left(z_s^{\text{hyp}} - z_s^{\text{dep}}\right) * \frac{1}{1+\exp(-10*(h[t-1]-0.5))} \\ h_s[t] & = z_s \odot h_s[t-1] + (1-z_s) \odot h[t-1] \\ s_{\text{out}}\,[t] & = \text{ReLU}(h[t]) \end{cases} \tag{4.5}$$

$$\begin{cases} i[t] & = \alpha i[t-1] + W_s s_{\text{in}}\,[t] \\ x[t] & = \rho \cdot \tanh\left(\frac{i[t]}{\rho}\right) \end{cases} \tag{4.6}$$

In the equations above we can see several changes with respect to the BRC cell :

- **Introduction of a new variable** H_s: This variable is responsible for the slow negative feedback observed in excitability.
- **Introduction of the** r **and** r_s **gates**: r takes the place of the a variable in the BRC and is now fixed to $r = 2$ in order to be permanently bi-stable and create positive feedback. r_s on the other hand modulates the strength of the slow feedback on the fast variable h of the system. In practice, it is a constant fixed to $r_s = -7$ to have negative feedback.
- **Introduction of the** z **and** z_s **gates**. z modulates the fast feedback speed and takes the place of the c gate in the BRC. To obtain the desired spikes it is set to its maximum speed with $z = 0$. z_s on the other hand modulates the speed of the negative feedback. It is a function of h shaped by the constants z_s^{hyp}, z_s^{dep}, and the function $\frac{1}{1+\exp(-10*(h[t-1]-0.5))}$. This allows the negative feedback to be slow at low values of h values and larger for high values of h. This assists in the recreation of sub-threshold dynamics as well as having a fast down-stroke like it is observed in biological neurons. In order to obtain spikes similar to biologically generated ones, the z_s constants are fixed to $z_s^{hyp} = 0.9$ and $z_s^{dep} = 0$.
- The output of the neuron is the fast variable h. However, it goes through a $ReLU(\cdot)$ function in order to cut the sub-threshold dynamics from propagating through the network as only spike communication is desired.
- **A new learnable parameter** b_h **is introduced** to modulate the firing threshold of the neuron.

Figure 4.7 shows the simulation of the SRC equations using the PyTorch[70] Machine learning framework. The two first panes show how the inputs are integrated over time using the exponential decay input synapse. In the two next panes, we can see the fast and slow feedback that creates the spiking behaviour. Finally, the fast dynamics of the neuron are filtered using a $ReLU(\cdot)$ function to remove the sub-threshold dynamics from the output and only communicate spikes to the post-synaptic neurons.

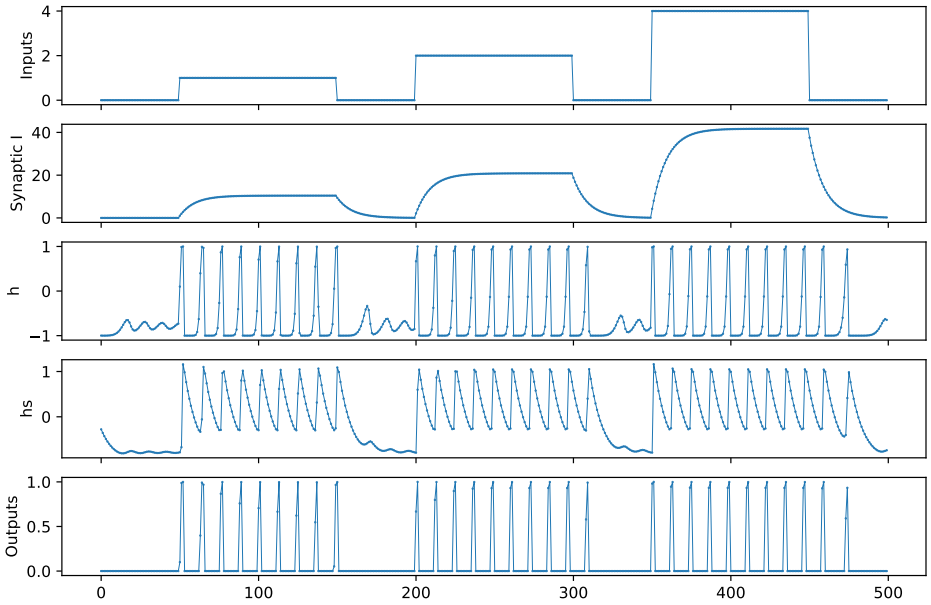


Figure 4.7: Simulation of an SRC cell implemented using the PyTorch framework. In these simulations, the dynamics of each component of the neuron have been separately plotted. Namely, one can see the synaptic input, the SRC fast and slow dynamics, as well as the output of the cell as seen from the postsynaptic cell.

In their work, De Geeter et al. show that this new architecture can create a spiking neural network that learns through back-propagation without the need for approximations, unlike the previously discussed surrogate gradients. Furthermore, from the equations above, it can be seen that not only do networks learn at the synapse as in traditional SNNs but they can also modulate the neurons' sensitivity itself through b_h giving one more degree of freedom to the SNN. This novel approach to SNNs was tested on the MNIST benchmark and obtained 98% accuracy which is close to state-of-the-art performances with relatively small SRC-based spiking neural networks that consist of 1 to 3 hidden layers of 512 neurons per layer.

Chapter 5

Designing RNN Cells From Known Neuron Dynamics

The SRC paves the way towards a new way of creating SNNs that learn within the widespread RNN framework learning through the successful back-propagation algorithm. However, there are several improvements that can be made to the SRC. Notably, the SRC is not modulable, meaning that it only performs one type of spiking pattern. Furthermore, bursting is also a capability that has been observed across the brain that is lacking from the SRC neuron. In order to expand the work done with the SRC, in this chapter we will explain the tools that can be used to analyse the dynamics of RNN cells. We will then use these tools to analyse the behaviour of the SRC. Finally, we will propose a cell that expands on the SRC and adds modulation capabilities. The dynamics of this expanded cell will also be analysed and compared to known neuron models capable of the same modulations.

5.1 RNNs Define Discrete Difference Equations of Continuous Differential Equations

With complex partial differential equation(PDE) systems, it is often challenging to find analytical solutions to the system. It is more convenient to solve the equations using numerical methods that approximate the solution. In numerical analysis, the Finite Difference Method(FDM) approximates derivatives as finite differences, both in the spatial and temporal domains. The domains are first discretised into a set of coordinates or time steps of finite length and duration. Then, with the finite difference method, a set of PDEs can be defined as a set of linear equations that is solved with classic linear algebra matrix and vector operations.

5.1.1 The Finite Difference Method (FDM)

The FDM is based on the Taylor expansion of a function. For a function $f(\cdot)$ that is at least n times differentiable, the Taylor expansion is given in equation 5.1, where $O(\Delta x)^{n+1}$ is the upper bound of the approximation error.

$$f(x+\Delta x) = f(x) + \frac{f'(x)}{1!}\Delta x + \frac{f^{(2)}(x)}{2!}\Delta x^2 + \cdots + \frac{f^{(n)}(x)}{n!}\Delta x^n + O(\Delta x)^{n+1} \tag{5.1}$$

Using the Taylor expansion the FDM approximates a derivative by the difference $f'(x) \approx f(x+\Delta x) - f(x)$. More precisely by rearranging equation 5.1 we can obtain the *Forward Difference Scheme* of the FDM which is shown in equation 5.2 here bellow.

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + O(\Delta x) \tag{5.2}$$

Using the forward difference scheme one can go back and forth between an RNN update equation and the differential equation that it simulates.

5.1.2 Applying the FDM on the SRC to Obtain its Equivalent Differential Equations

In order to show the equivalence between an RNN and a set of differential equations we can apply the forward difference scheme on the SRC to obtain the equivalent differential equation that is modelled by the RNN cell.

In equation 5.3 we rewrite the SRC equations shown in 4.5 replacing the constants $r = 2$, $z = 0$, $z_s^{hyp} = 0.9$, and $z_s^{dep} = 0$ by their numerical values for conciseness.

$$\begin{cases} h[t] & = \tanh\left(x[t] + 2h[t-1] + r_s \odot h_s[t-1] + b_h\right) \\ h_s[t] & = z_s(h[t-1]) \odot h_s[t-1] + (1 - z_s(h[t-1])) \odot h[t-1] \\ z_s(h) & = 0.9 - 0.9 * \frac{1}{1+\exp(-10*(h-0.5))} \end{cases} \tag{5.3}$$

In equation 5.3 it can be seen that $h[t]$ and $h_s[t]$ use difference equations in the time domain as the value at time step t depends on values at time step $t-1$. Let's begin with the $h[t]$ equation and relate it to the forward difference scheme from equation 5.2 and develop the equivalence :

$$\begin{aligned} & h(t+\Delta t) = h(t) + h'(t)\Delta t \\ & h[t+\Delta t] = zh[t] + (1-z)\tanh\left(x[t] + 2h[t] + r_s h_s[t] + b_h\right) \\ & \text{set } \Delta t = (1-z), \text{ with } z = 0 \Rightarrow \Delta t = 1 \\ & \Rightarrow h(t+1) = h(t) + h'(t) = \tanh\left(x(t) + 2h(t) + r_s h_s(t) + b_h\right) \\ & \Rightarrow h'(t) = h(t+1) - h(t) = \tanh\left(x(t) + 2h(t) + r_s h_s(t) + b_h\right) - h(t) \end{aligned} \tag{5.4}$$

The same can be done with the h_s equation to obtain the system of differential equations 5.5 that defines the SRC continuous time system.

$$\begin{cases} h'(t) = \tanh\left(I_{ext}(t) + rh(t) + r_s h_s(t) + b_h\right) - h(t) \\ h'_s(t) = z_s(h(t)) \odot h_s(t) + (1 - z_s(h(t))) \odot h(t) - h_s(t) \\ z_s(x) = 0.9 - 0.9 * \dfrac{1}{1 + \exp(-10 * (x - 0.5))} \end{cases} \tag{5.5}$$

5.2 Analysis of the SRC neuron

Once we have the differential equations defining the SRC neuron, the system can be simulated as shown in figure 5.1 and the non-linear dynamics of the system can be studied using the same tools used to discuss the different types of excitability chapter 2.

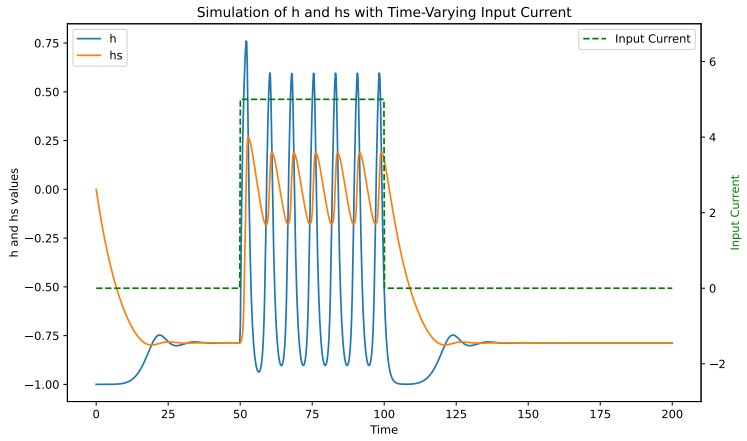


Figure 5.1: Simulation of the continuous system defined by the SRC RNN cell equations for a step input current.

We can start to analyse the cell by observing the FI curve shown in figure 5.2. We can see that the cell has a significant step in the obtained frequencies at $I_{app} = 1$. This discontinuity in the FI curve is conducive to type 2 excitability. Further increasing I_{app} increases the spiking frequency slightly until a sudden drop to 0 Hz due to the cell reaching saturation.

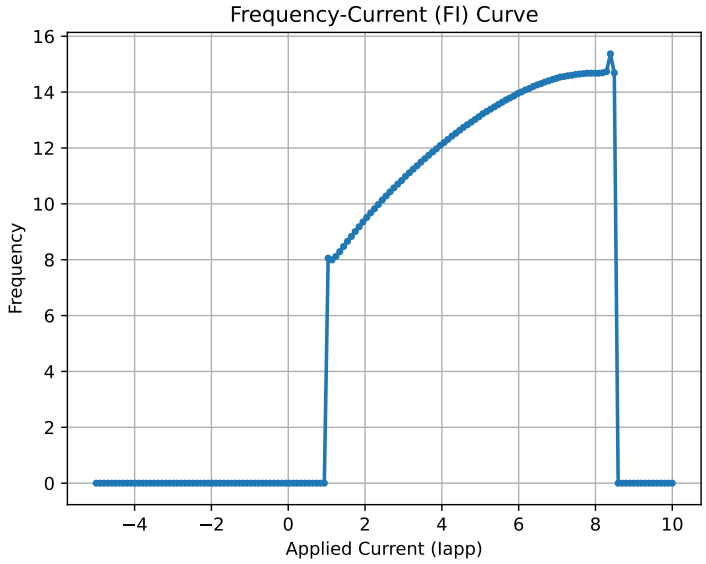


Figure 5.2: Frequency-Current curve for the SRC cell. A discontinuity in the frequencies can be seen indicating type 2-like behaviour. The frequency drop above a certain input intensity is due to cell saturation.

The mechanism through which excitability is obtained can be studied further through the use of phase portraits. Figure 5.3 shows the phase portrait at 3 different values of input currents. On the left, we see that there is no cycle in the trajectory indicating a stable fixed point. The middle phase portrait shows damped oscillations indicating that the stable point is destabilising as I_{app} is increased. We can also see that the effect of I_{app} is to slide the z shaped h-nullcline to the right, effectively sliding the fixed point along the

h-nullcline. On the rightmost phase portrait, the fixed point is fully destabilised and a stable limit-cycle is formed around it. The sequence of events describes a Hopf bifurcation which is the mechanism that creates type 2 spiking, further confirming that the SRC cell only creates type 2 excitability.

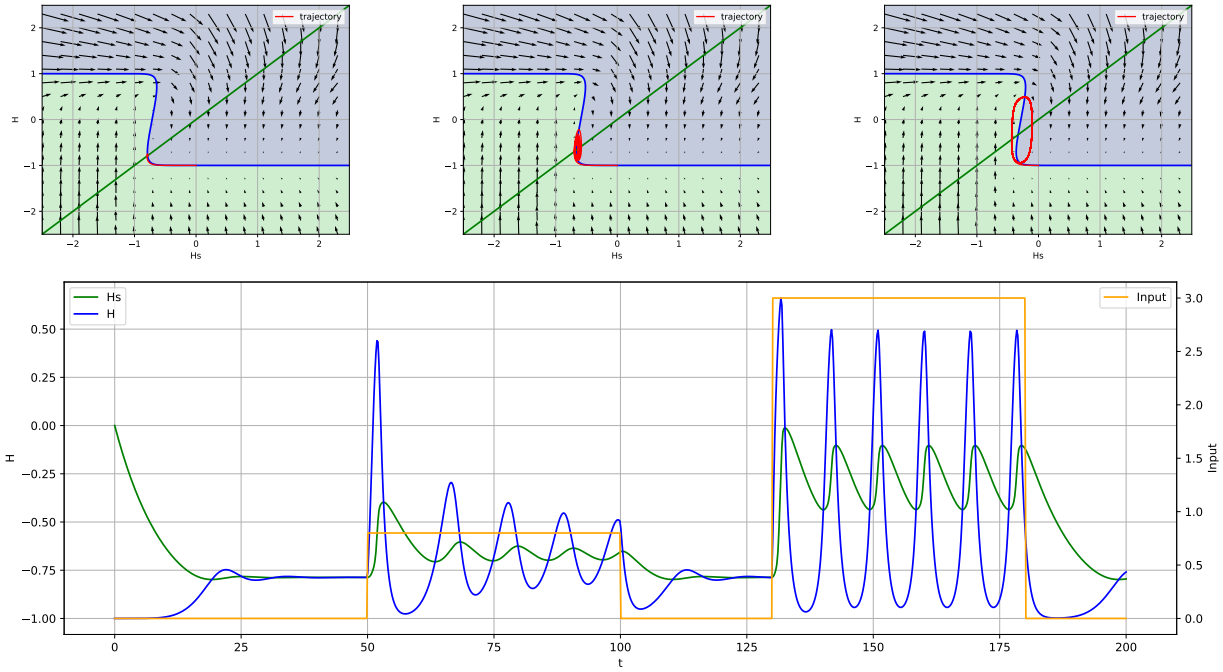


Figure 5.3: Phase portraits and voltage trace of the SRC cell at I_{app} = 0, 0.85, and 3 respectively.

5.3 Design of a Modulable Spiking Recurrent Cell (mSRC)

In the previous section a limiting factor of the SRC cell was uncovered, that is, a lack of neuromodulation capabilities. The cell can only modulate its sensitivity to input by modulating the firing threshold, it cannot change its response to the input like biological neurons can by changing their excitability type. To overcome this limitation, in this thesis, we take the SRC neuron as a base and draw inspiration from the extension of the Fitzhugh-Nagumo model in [21],[22] and [23] to give neuromodulation capabilities to the SRC cell, creating the modulable Spiking Recurrent Cell (mSRC).

5.3.1 Design

In order to obtain modulation from the classic Fitzhugh-Nagumo, a transcritical bifurcation is introduced into the phase portrait. This bifurcation is described as the *organising centre* in neuronal excitability. In order to create the transcritical bifurcation, they introduce symmetry by creating a reflection of the V-nullcline along the vertical axis. This is done by making the slow feedback go through a symmetric function like the squared function. As a reminder, equation 5.6 shows the equation of the Fitzhugh-Nagumo model and 5.7 shows its modulable extension.

$$\begin{cases} \dfrac{dV_m}{dt} = V_m - \dfrac{V_m^3}{3} - W + I_{ext} \\ \tau \dfrac{dW}{dt} = (V_m + a - bW) \end{cases} \tag{5.6}$$

$$\begin{cases} \dfrac{dV_m}{dt} = V_m - \dfrac{V_m^3}{3} - (W + W_0)^2 + I_{ext} \\ \tau \dfrac{dW}{dt} = (V_m + a - bW) \end{cases} \tag{5.7}$$

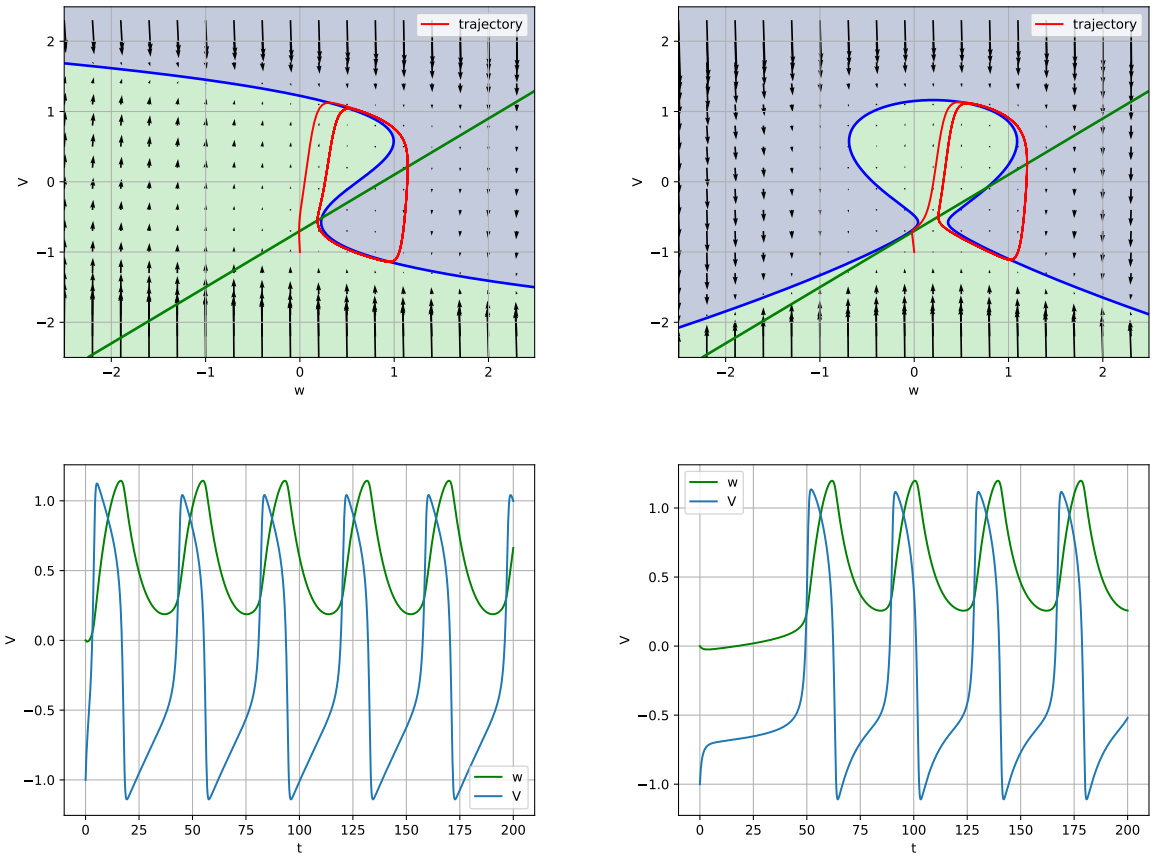


Figure 5.4: Phase portraits(top) and voltage traces(bottom) of the original FitHugh-Nagumo model on the left and its extension by adding the reflection of the V-nullcline on the right. We can see in this case that the simple addition of the reflection decreases the frequency and introduces a spiking delay in the voltage traces.

Figure 5.4 shows the FitzHugh-Nagumo on the left, and on the right, the effect of adding the squared function around the slow feedback in the fast dynamics. From a pure feedback point of view, adding the squared function allows modulating the negative feedback strength at the spiking threshold by modulating its minimum value in the fast variable. This creates 3 different cases responsible for the 3 different studied firing patterns.

- If the feedback is negative at the threshold, it promotes a competition between the fast and slow variable. High input is necesary to overcome this competition and it results in rapid spiking characteristic

of type2 firing.

- If the feedback is insensitive near the threshold we can obtain arbitrarily slow spiking, characteristic of type1 excitability.
- With positive feedback near the threshold, there is cooperation between the fast and slow variables of the system, effectively promoting the creation of subsequent spikes after the first one. This characterises the hysteresis behaviour of type 2*.

From the Fitzhugh-Nagumo model to the SRC model we can see that the V is the fast variable represented as h in the SRC and w is the slow variable that corresponds to the h_s variable in the SRC. The main difference between the two models is the chosen non-linearity to create a pitchfork bifurcation around the fast variable. In the phase portrait, this non-linearity gives the the z shape to the V nullcline. In the Fitzhugh-Nagumo model, the cubed function is chosen, however, the SRC uses the $\tanh(\cdot)$ function. From a systems point of view, it does not change the bifurcations and the behaviours are qualitatively identical. This change was made for its better Machine Learning properties. Indeed, the $\tanh(\cdot)$ function has gradients that are bounded by $[0,1]$, this creates a vanishing gradients problem but alleviates the exploding gradients. On the other hand, the derivative of the cubic function x^3 is $\frac{dx^3}{dx} = 3x^2$ which not only suffers from vanishing gradients when x is small but it also exacerbates the exploding gradients problem for large x.

In order to obtain the sought-out symmetry in the phase portrait of the SRC model we can use the same strategy as with the extension of the Fithugh-Nagumo model and wrap the slow feedback in a squared function with an h_{s0} parameter to modulate the behaviour of the cell. The modulable SRC(mSRC) system equations can be seen here below in equation 5.8. In figure 5.5 we can see the simulation of a trajectory along with the phase portrait corresponding to the new equations on the right compared to the original SRC equations on the left.

$$\begin{cases} h'(t) = \tanh\left(I_{ext}(t) + rh(t) + r_s(h_{s0} + h_s(t))^2 + b_h\right) - h(t) \\ h'_s(t) = z_s(h(t)) \odot h_s(t) + (1 - z_s(h(t))) \odot h(t) - h_s(t) \\ z_s(x) = 0.9 - 0.9 * \dfrac{1}{1 + \exp(-10 * (x - 0.5))} \end{cases} \tag{5.8}$$

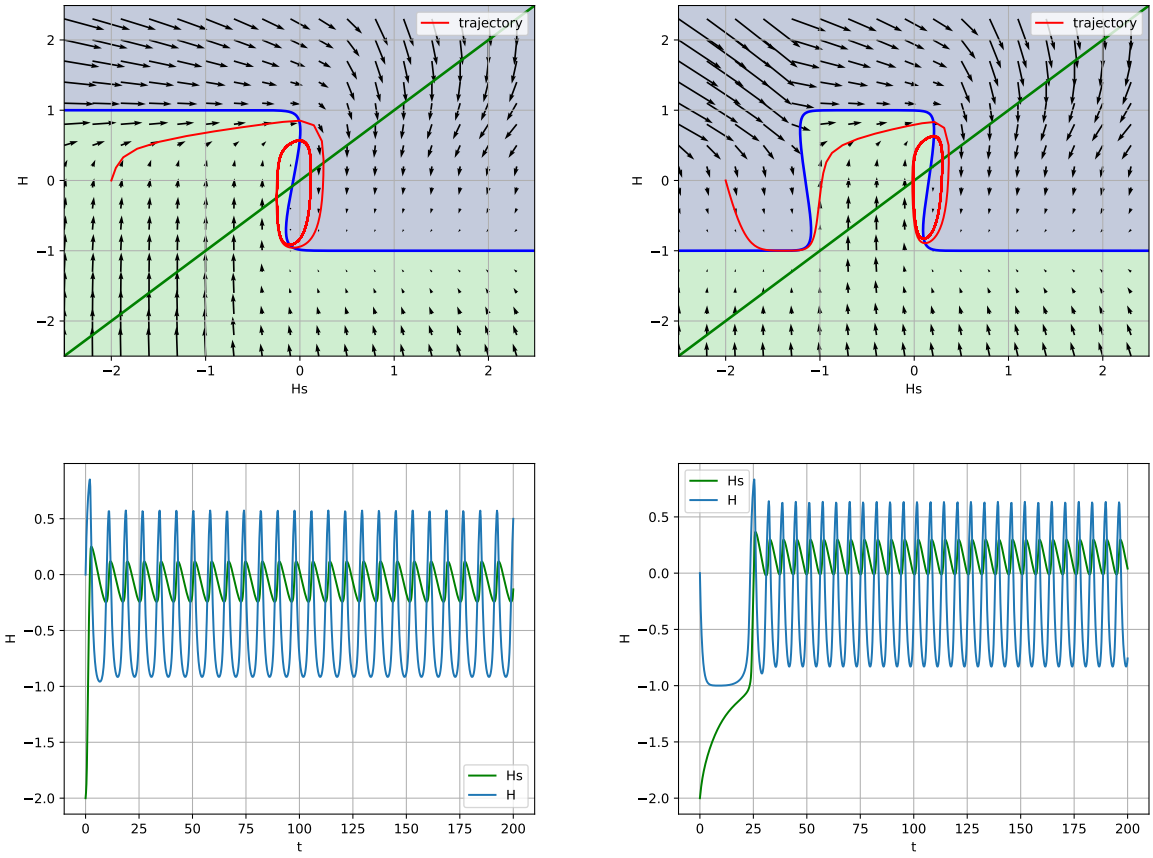


Figure 5.5: Spiking phase portraits and traces of the SRC model on the left and its extended version on the right.

In the previous simulations, we can see that both cells do not reach the upper part of the V nullcline, meaning that spikes are not all-or-none events. This problem is caused by 2 main issues. The first is an inadequate $z_s(x)$ equation which modulates the speed of the slow h_s variable as a function of the value of the fast h variable. Indeed, having the speed of h_s that increases prematurely will stop the upstroke of the spike before reaching its maximum by increasing the negative feedback too soon. This hinders the *all or none* nature of spikes and makes their size sensitive to input strength. This is an undesirable behaviour of the model as we do not want input strength to be encoded in the amplitude of the spike. To solve this issue we can reduce the speed of h_s for a value of h far from its maximum value of 1. In figure 5.6 we can see the graphs of $z_s(h)$. On the left, we can see the original $z_s(h)$ as proposed for the SRC. On the right, we can see a shifted version of the function with a faster change in speed. The second source of the issue comes from the weak positive feedback that does not create a fast enough upstroke. This can be solved by increasing the positive feedback term $r = 2$ to $r = 4$.

In figure 5.7 we can see the effect of these changes on the mSRC cell. On the right with the modified z_s and increased r we can see an improved all or none characteristic in the trajectories of the spikes compared to the base mirrored SRC on the left. The resulting equations from these modifications can be seen in equation 5.9. These equations will be the ones used for the rest of this chapter.

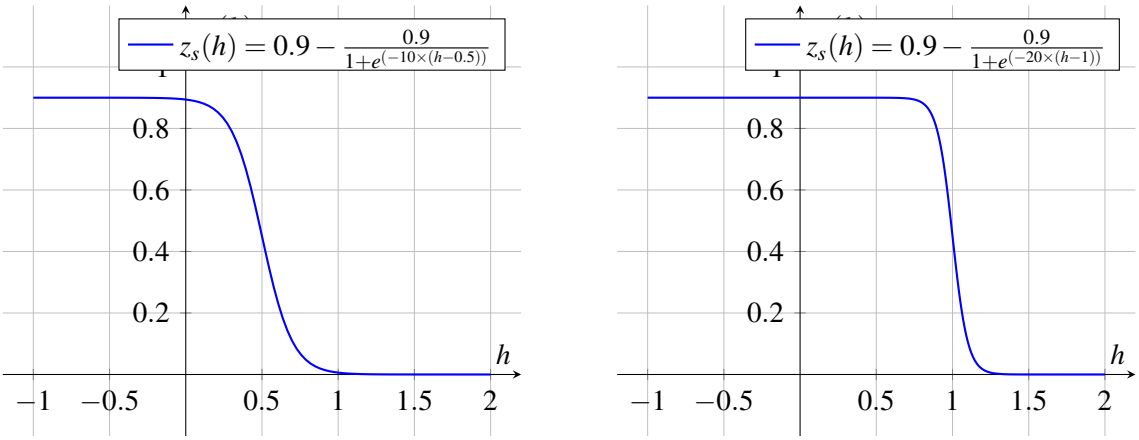


Figure 5.6: Plots of the slow feedback speed modulating functions. On the left we can see the original SRC equation and on the right the modified version for the mSRC simulations.

$$\begin{cases} h'(t) = \tanh\left(I_{ext}(t) + 4 \cdot h(t) - 7 \cdot (h_{s0} + h_s(t))^2 + b_h\right) - h(t) \\ h'_s(t) = z_s(h(t)) \odot h_s(t) + (1 - z_s(h(t))) \odot h(t) - h_s(t) \\ z_s(x) = 0.9 - 0.9 * \dfrac{1}{1 + \exp(-20 * (x-1))} \end{cases} \tag{5.9}$$

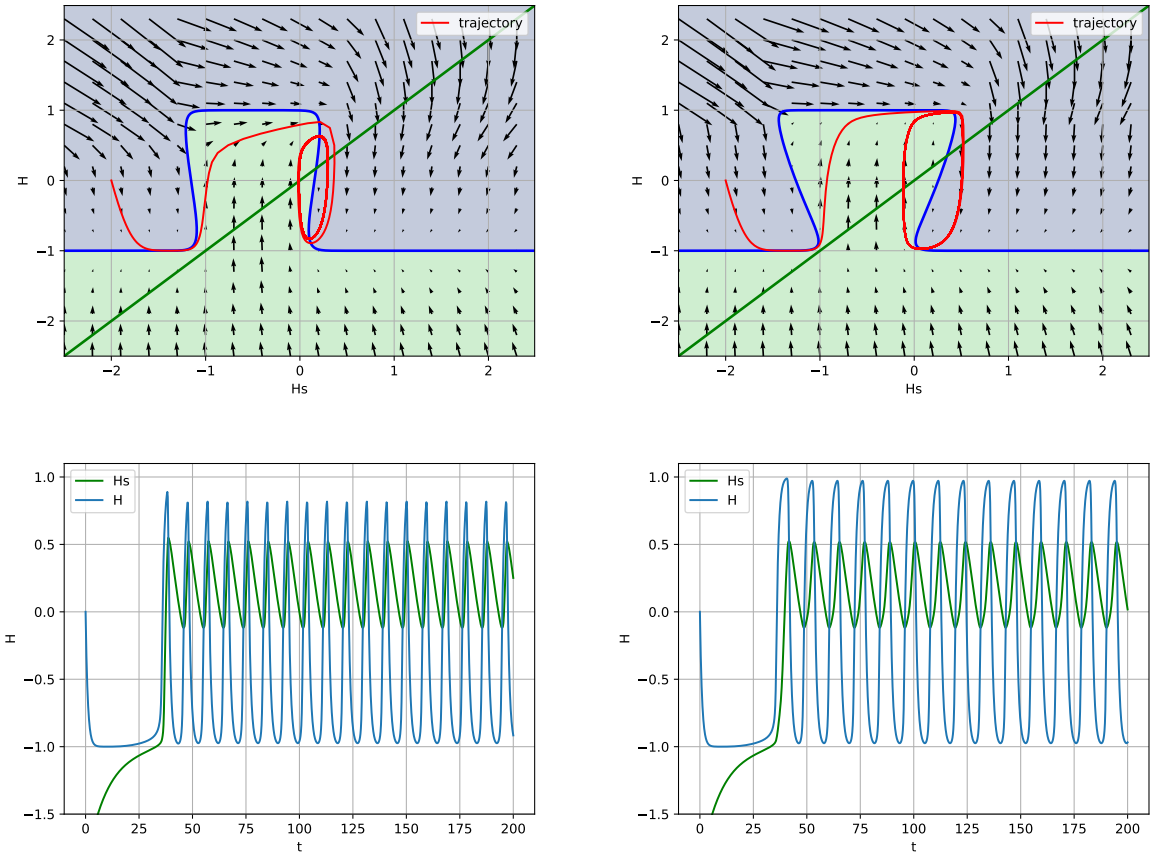


Figure 5.7: Spiking phase portrait and voltage traces of the base mSRC model on the left and its improvement with $r = 4$ and modified $z_s(h)$ on the right.

5.3.2 Analysis

Having obtained the mSRC equations, the qualitative behaviour, as well as the phase portraits of the systems can be analysed. In this section, we discuss the behaviours obtained with the mSRC through, FI curves and bifurcation analysis in an attempt of reproducing the 3 different spiking behaviours that were sought out in this thesis.

a Qualitative Behaviour

Figure 5.8 shows 3 heatmaps corresponding to the mSRC neuron. On the left, the FI heatmap shows that for values of h_{s0} close to 0.9, the FI curve shows a larger frequency range, indicating that this could be the narrow type 1 firing region surrounded by type 2-like behaviour. Figure 5.9 takes slices of the FI heatmap at 3 different values to more clearly show a the evolution of the behaviour when $h_{s0} = 1.5, 0.9,$ and 0.5. The left and bottom heatmaps show the firing latency from rest, and the hysteresis size between the step-up and step-down FI curves respectively. The rightmost FI curve in figure 5.9 also shows the difference in frequency from step up and step down FI curves that take place at low values of h_{s0} . From these figures, we can see that for small values of $h_{s0} \approx 0.5$ we obtain behaviours that are characteristic of type 2* firing. We also see in the FI heatmap that this type 2*-like region has lower current saturation values. Finally, this leaves the region with high values of h_{s0} as a candidate for true type 2 firing. In the following subsections,

we will study the phase portraits in these regimes and compare them to the previously seen phase portraits of the reduced Connor-Stevens model.

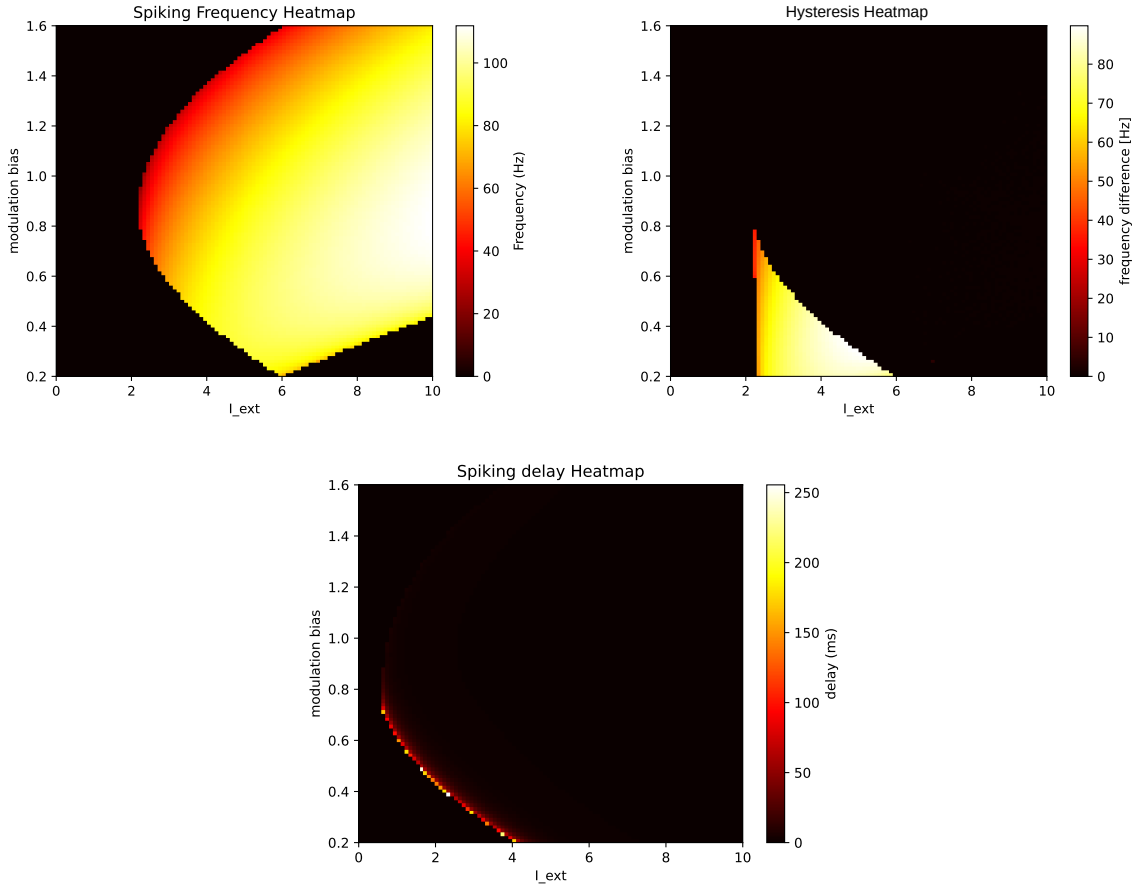


Figure 5.8: Heatmaps corresponding to the different mSRC h_{s0} modulation values. On the left, the FI heatmap can be seen showing a decrease in discontinuity of frequencies around $h_{s0} = 0.9$. On the right the spiking delays heatmap shows an increase in delays for lower values of h_{s0}. At the bottom we can see the increasing hysteresis range as h_{s0} decreases.

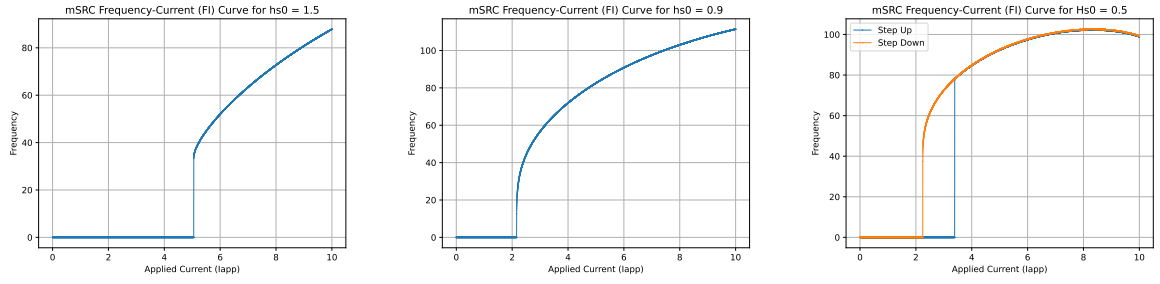


Figure 5.9: FI curves computed at increasing values of h_{s0}. From left to right $h_{s0} = 1.5, 0.9$, and 0.5. We can see how the discontinuous characteristic decreases with $h_{s0} = 0.9$ and then increases again with lower values. With $h_{s0} = 0.5$ we can see that the step up FI curve behaves much like type2 while the step down has a larger frequency range effectively creating hysteresis.

b Type 2 : $h_{s0} = 1.5$

In figure 5.10 we can see the phase portraits of the mSRC neuron with a modulation bias $h_{s0} = 1.5$ at subthreshold stimulation on the left, at the threshold in the middle, and at suprathreshold stimulation on the right. From these phase portraits, we can see the merging of the two h-nullclines in the system as the stimulation is increased, reminiscent of the extended Fitzhugh-Nagumo model and the reduction of the Connor-Stevens model. Furthermore, there is always a single fixed point in the system, on the left we can see it is stable. Increasing stimulation destabilises it, creating oscillations at the threshold, and finally creating a stable limit cycle at suprathreshold stimulation. This behaviour describes the Hopf bifurcation confirming type 2 excitability for elevated value of h_{s0}.

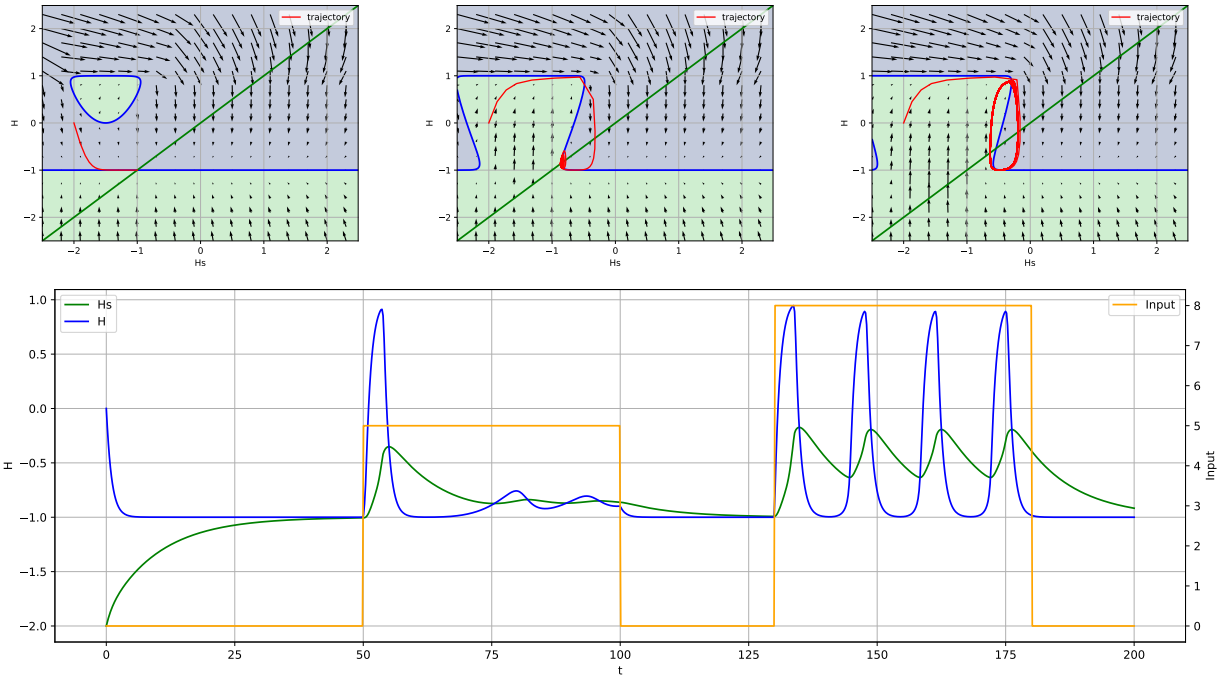


Figure 5.10: Phase portraits and voltage trace of the mSRC neuron in type 2 regime with $h_{s0} = 1.5$ at $I_{ext} = 0, 5$, and 8 respectively.

c Type 1 : $h_{s0} = 0.9$

As it can be seen from figure 5.10 to 5.11, decreasing h_{s0} to 0.9 slides the V nullclines to the right. At this value of h_{s0}, we can see in figure 5.11 that increasing stimulation fuses the 2 h nullclines close to the fixed point. Indeed, the fusing of the 2 nullclines approximately corresponds to the destabilising of the fixed point. This creates a bottleneck between the left and right parts of the h nullclines and also destabilises the fixed point through a SNIC bifurcation. These are characteristics that also described type 1 excitability in the reduced CS model.

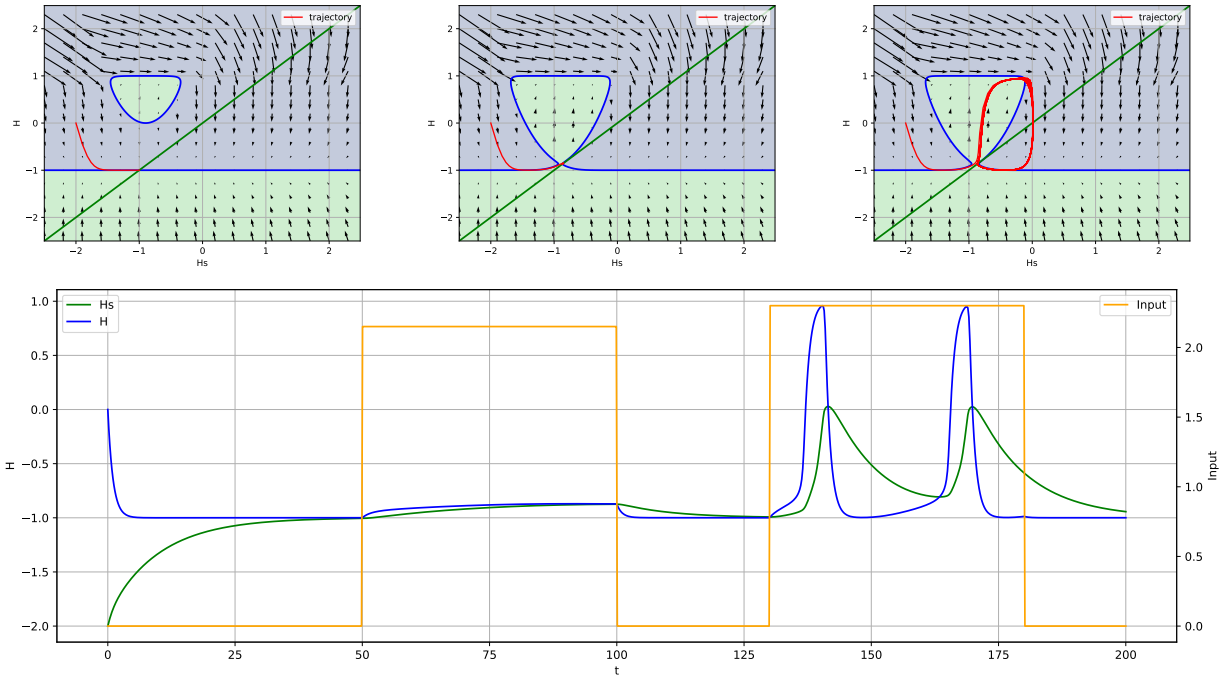


Figure 5.11: Phase portraits and voltage trace of the mSRC neuron in type 1 regime with $h_{s0} = 0.9$ at $I_{ext} = 0, 2.15$, and 2.3 respectively.

d Type 2* : $h_{s0} = 0.5$

Further shifting the h nullcline to the right with $h_{s0} = 0.5$, we obtain the phase portraits in figure 5.12. In the middle phase portrait, we can see that after the 2 h nullclines fuse there is a coexistence of 2 fixed points. The fixed point on the left is stable and we can see through the vector field that it attracts the trajectories that are in the left region of the phase portrait. Further increasing stimulation destroys the leftmost fixed point through a saddle-node bifurcation, only leaving the rightmost unstable fixed point and its stable limit cycle to attract the trajectories. To terminate the trajectory after the onset of the limit cycle, stimulation has to be decreased until the leftmost fixed point is created again and the trajectory crosses the separatrix between the stable limit cycle and the stable fixed point. This is the source of the observed hysteresis and is also the same mechanism observed in the type2* excitability of the Connor-Stevens model.

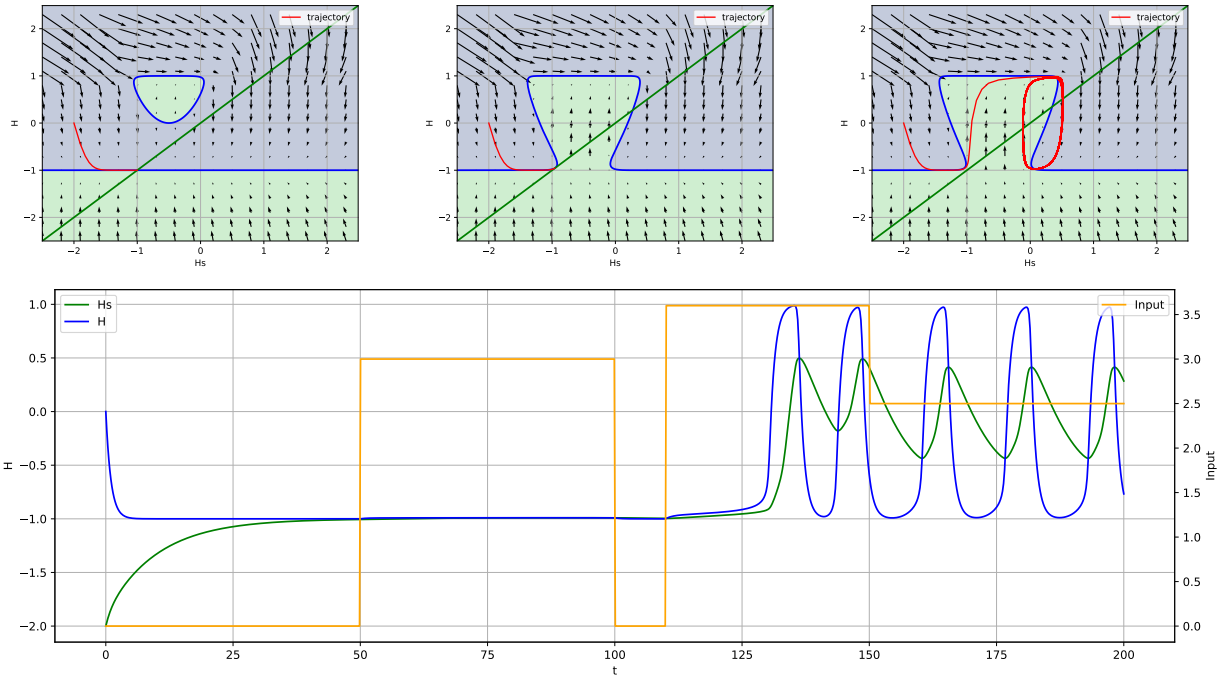


Figure 5.12: Phase portraits and voltage trace of the mSRC neuron in type 2* regime with $h_{s0} = 0.5$ at $I_{ext} = 0, 3, 3.6$,and in the voltage trace, 2.5 again. The voltage trace shows the two main characteristics of type2*, the spiking delay at $I_{ext} = 3.6$ and the hysteresis as $I_{ext} = 2.5$ is above the threshold after the onset of the spikes but not from quiescence.

5.4 Adding bursting capabilities to the cell, the BSRC

In order to complete the parallel with the extended Fitzhugh-Nagumo model discussed in Chapter 2, we can also add an ultra-slow timescale negative feedback to the mSRC neuron in the same manner to obtain bursting capabilities. The equations of the Bursting Spiking Recurrent Cell can be observed in 5.10. The simulation of these equations in the bursting regime can be seen in figure 5.13. Even though bursting is out of the scope of this thesis, it is important to mention that this third equation does not only bring bursting capabilities to the cell. It has been shown [71] that ultra-slow feedback also assists in the broadening of the type1 behaviour region while also linearising the FI curve.

$$\begin{cases} h'(t) = z \cdot h(t) + (1-z) \cdot \tanh(x(t) + r \cdot h(t) + r_s \cdot (h_{s0} + h_s(t))^2 + b + r_u \cdot h_u(t)) - h(t) \\ h'_s(t) = z_s(h) \cdot h_s(t) + (1 - z_s(h(t))) \cdot h(t) - h_s(t) \\ h'_u(t) = z_u \cdot h_u(t) + (1 - z_u) \cdot (h(t) + 0.5) - h_u(t) \\ z_s(x) = 0.9 - 0.9 \dfrac{1}{1 + \exp(-20 \cdot (x-1))} \end{cases} \tag{5.10}$$

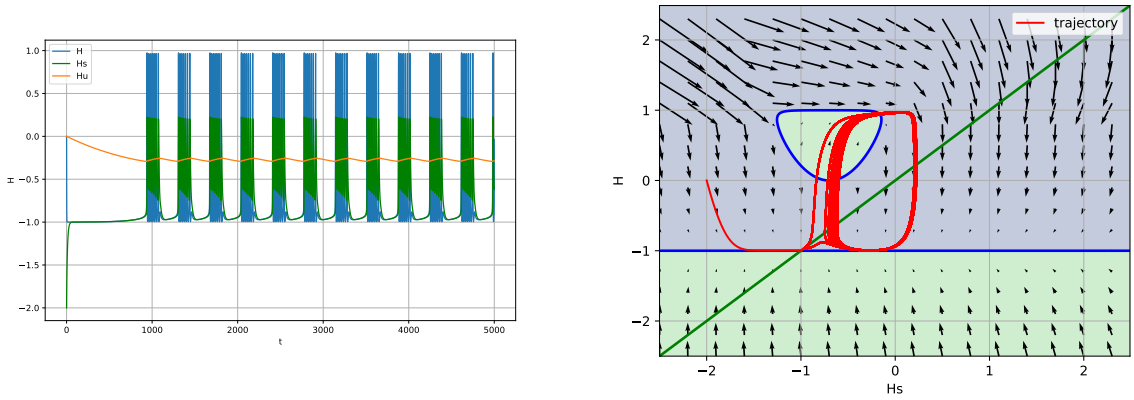


Figure 5.13: Simulation of the BSRC equations 5.10 with the following constant values: $x = 0$, $r = 4$, $r_s = -7$, $r_u = -8.5$, $z = 0$ $z_u = 0.999$, $b = 0$, and $h_{s0} = 0.7$. On the left we can see the variation of the 3 variables with time, and on the right a phase portrait of the fast and slow equations with a fixed to zero ultra-slow variable. The trajectory over the fast and slow variables is overlayed in red.

5.5 Does adding a synapse model to the input alter the system?

When simulating a network of spiking neurons, not only the behaviour of the individual neurons has to be taken into account, but their connections also play an essential role in the computations performed by the network. As previously discussed, there are 3 main types of synapse models used in SNNs: static, exponential, and double exponential models. Static models are a simple identity function with respect to the dynamics of the presynaptic neuron which will not affect the behaviours of the postsynaptic neuron. However, the two other models are systems on their own and coupling their dynamics to a neuron can alter the qualitative properties of a neuron model.

Equations 5.11 integrate an exponential decay synapse to the input of the mSRC neuron as the one described for the SRC neuron with a decay factor α.

$$\begin{cases} x'(t) = \alpha x(t) + I_{ext}(t) - x(t) \\ h'(t) = \tanh\left(x(t) + 4 \cdot h(t) - 7 \cdot (h_{s0} + h_s(t))^2 + b_h\right) - h(t) \\ h'_s(t) = z_s(h(t)) \odot h_s(t) + (1 - z_s(h(t))) \odot h(t) - h_s(t) \\ z_s(x) = 0.9 - 0.9 * \dfrac{1}{1 + \exp(-20 * (x - 1))} \end{cases} \tag{5.11}$$

Figures 5.14 and 5.15 show the heatmaps for the new full system with the integration of a synapse of decay factor $\alpha = 0.9$ and $\alpha = 0.5$ respectively. As can be seen in those figures, the integration of the synapse greatly modifies the behaviours of the neuron. Indeed, the synapse acts as an integrator of inputs on its own, amplifying the input and reducing the effect of type 1 behaviour. It also decreases the input amplitude required for saturation. Furthermore, the spiking delay and hysteresis heatmap are very different when a synapse is introduced. Even though memory and spiking delay can be observed at all modulation values, it is important to note that the mechanisms through which they are created are different. Indeed, leaky synapses have memory and delay on their own as they lag behind the input, meaning that the current does not directly start or stop with the stimulation, this is why with the used protocol both are still detected. However, we can see that the frequency gap and delays are small with a synapse as it is no longer bistability but a lagged response to a change in input that is mainly observed in these heatmaps. It

is therefore important to keep in mind that the synapses and their parameter are also central to the overall network behaviour and performance of an SNN.

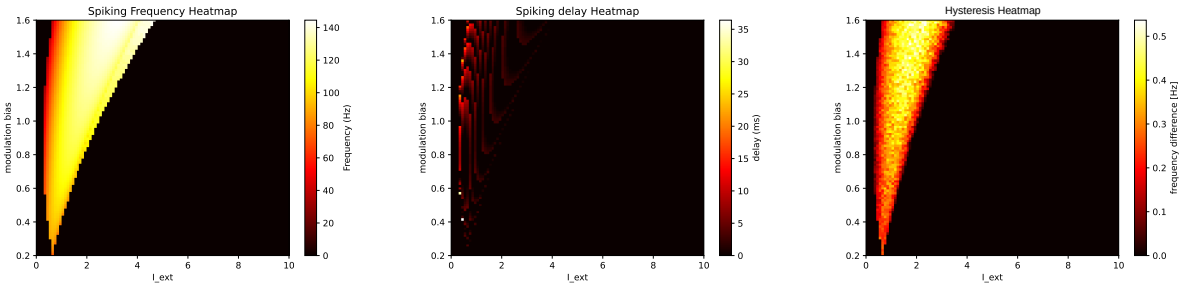


Figure 5.14: Heatmaps corresponding to the different mSRC h_{s0} modulation values with an exponential synapse at input with $\alpha = 0.9$. We see the FI, spiking delay, and hysteresis amplitude heatmaps from left to right.

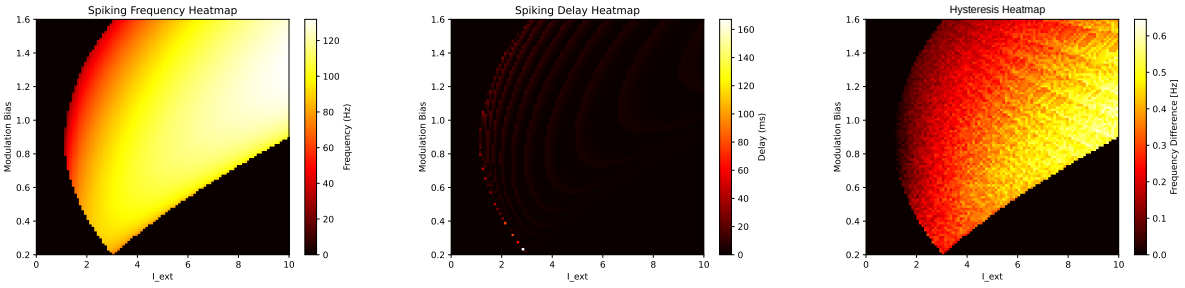


Figure 5.15: Heatmaps corresponding to the different mSRC h_{s0} modulation values with an exponential synapse at input with $\alpha = 0.5$. We see the FI, spiking delay, and hysteresis amplitude heatmaps from left to right.

5.6 Using the FDM to go back to an RNN equation

Using the same derivation as before in equations 5.4 we can simply go back to an update equation by adding a variable back to its differential equation. The update equations for the mSRC and BSRC can be seen in 5.12 and 5.13 respectively.

$$\begin{cases} h[t+1] = \tanh\left(x[t] + 4 \cdot h[t] - 7 \cdot (h_{s0} + h_s[t])^2 + b_h\right) \\ h_s[t+1] = z_s(h[t]) \odot h_s[t] + (1 - z_s(h[t])) \odot h[t] \\ z_s(x) = 0.9 - 0.9 \dfrac{1}{1 + \exp(-20 * (x-1))} \end{cases} \tag{5.12}$$

$$\begin{cases} h[t+1] = z \cdot h + (1-z) \cdot \tanh(x[t] + r \cdot h[t] + r_s \cdot (h_{s0} + h_s[t])^2 + b + r_u \cdot h_u[t]) \\ h_s[t+1] = z_s(h[t]) \cdot h_s[t] + (1 - z_s(h[t])) \cdot h[t] \\ h_u[t+1] = z_u \cdot h_u[t] + (1 - z_u) \cdot (h[t] + 0.5) \\ z_s(x) = 0.9 - 0.9 \dfrac{1}{1 + \exp(-20 \cdot (x-1))} \end{cases} \tag{5.13}$$

Figure 5.16 and 5.17 shows simulations of the cells implemented as RNN cells using the PyTorch[70] machine learning framework. As can be seen, the obtained behaviour is similar to the behaviour obtained in the continuous versions of the equations. In the mSRC simulation we can see the spiking delay with the intermediate input.

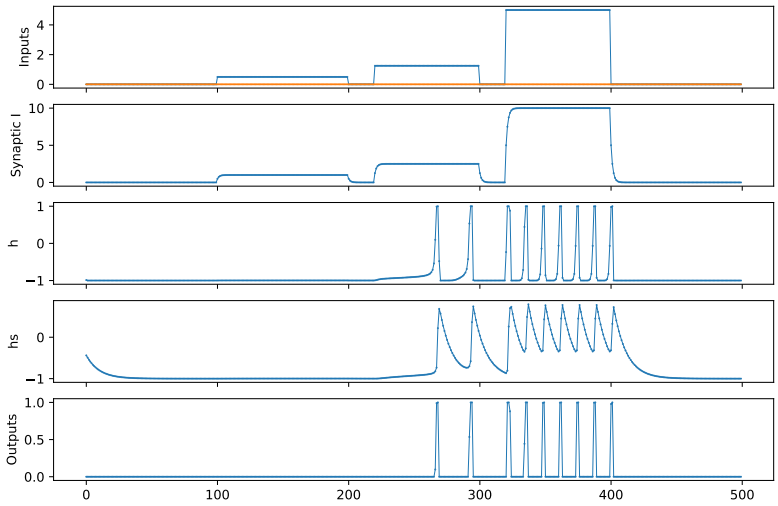


Figure 5.16: Simulation of an mSRC cell in type 2* regime $h_{s0} = 0.77$ with an exponential synapse of decay factor $\alpha = 0.5$ using the PyTorch framework.

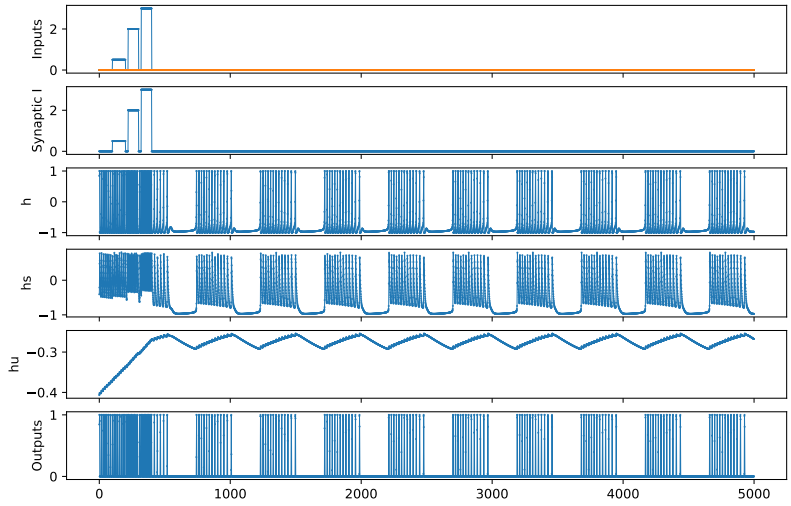


Figure 5.17: Simulation of a BSRC cell in bursting regime with the same parameters as in figure 5.13 with an exponential synapse of decay factor $\alpha = 0.5$ using the PyTorch framework.

5.7 Limits and considerations of the FDM

As every approximation and numerical method, the FDM has limitations and conditions to full-fill in order for the approximation to obtain the expected results. For this use case, there are 2 main criteria concerning the approximation error and the sampling frequency of the approximation.

The Finite Difference Method needs to be a stable scheme of simulation, meaning that the approximation error needs to be bounded. Intuitively, this criteria is automatically full-filled in our application as the cell uses gating to stay in the range $[-1,1]$. Therefore, no signal is amplified and the error stays bounded.

Using discrete time-step sizes to simulate the differential equation also brings sampling frequency limitations to the scheme. Indeed, sampling the signal at too low frequencies will distort the signal and cause aliasing. Aliasing in the context of neuron simulation would be particularly problematic as it could miss spikes or fuse them together in worst case scenarios. To avoid this, the sampling frequency $f_s = 1/\Delta t$ should be at least larger than twice the highest relevant frequency $f_{max} < 2f_s$ as stated in the Nyquist-Shannon theorem[72]. For our use case we know that $\Delta t = 1ms$, therefore we know that to prevent aliasing the dynamics of the cell should be contained in the $[0,200]Hz$ range. As we have seen in the previous heatmaps and FI curves, the spiking frequency of the cell lies within this range. However, an action potential has a higher frequency spectrum than its firing frequency due to the fast dynamics in the generating process, particularly at the peak of the spike. Therefore, we can expect spikes to have distorted dynamics close to their peak due to aliasing.

Chapter 6

Supervised Learning Tests

Having designed and analysed a Recurrent neural network cell from a system dynamics perspective, we can now integrate this cell into an ML environment a perform tests to evaluate the learning capabilities of the cell. In this chapter, we will provide the details of the machine learning implementation of an mSRC network to perform supervised learning tasks. We will evaluate each spiking regime's performance for the task's completion by forcing the modulation bias and learning other parameters. We will also assess the performance and analyse the network behaviour when modulation is introduced as a learning parameter.

6.1 Task Description

In order to assess the performance of the proposed cell and compare it to the SRC cell which is based on, image classification with the MNIST dataset will be used. However, as it is in digital form with pixel values ranging from 0 to 255, for its use in an SNN context, it has to be coded into spikes. To this end, we have decided to use both rate and latency coding approaches as described in the SRC tests to have a fair comparison.

In both approaches, each pixel is transformed into a sequence of 200 timesteps that can or not have a spike, respectively represented as values of 1 and 0. In the rate coding approach, the number of spikes is made proportional by sampling a probability distribution at every time step X_t with the probability of observing a spike $P(X_t = 1)$ that is proportional to the normalized intensity $I \in [0,1]$ of the pixel. With a proportionality factor $\alpha = 0.25$), the probability distribution is defined in equation 6.1. With this distribution, a black pixel $(I = 0)$ will never spike whereas a white pixel$(I = 1)$ has a 25% chance to spike at each time step.

$$P(X_t = 1|I = i) = i * \alpha \tag{6.1}$$

.

In the latency-based coding case, there is only one spike per pixel whose timing will depend on the intensity of the pixel. This makes this encoding highly sparser than the rate coding approach. For latency coding, brighter pixels will spike sooner than darker ones. The spike time t_{spk} of a pixel is defined as the duration needed by the potential of a linearised RC circuit of time constant τ to reach a threshold V_{th} if this circuit was powered by a current equivalent to the normalised pixel intensity I as shown in equation 6.2. In the performed tasks, $\tau = 10$ and $V_{th} = 0.01$ were used to replicate the SRC tests. Finally, the timing was normalised to fill the 200 timesteps of the sequence.

$$t_{spk} = min(-\tau(I-1), V_{th}) \tag{6.2}$$

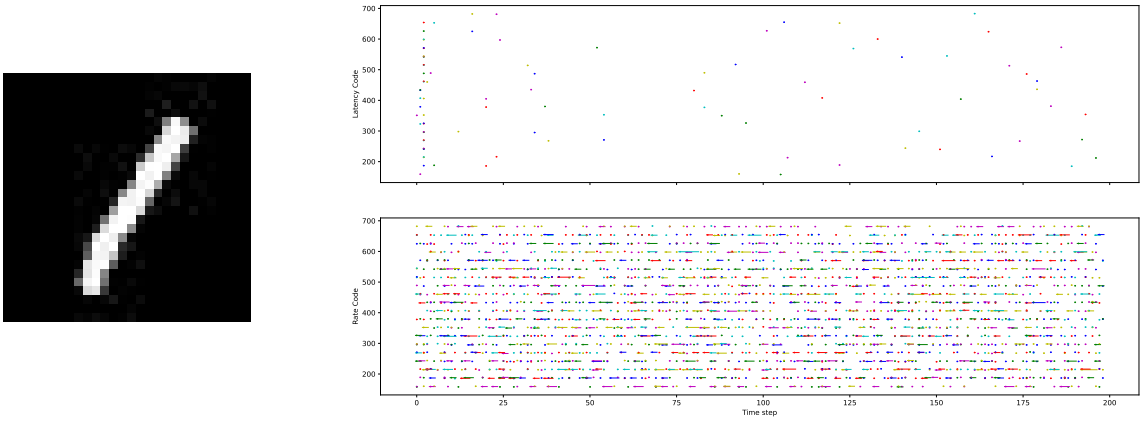


Figure 6.1: MNIST sample of the left and its respective spike rate and latency coding on the right.

Figure 6.1 shows rate and latency coding raster plot examples of an MNIST sample using the previously described approaches. Both were implemented using the SNNTorch library[73]. It can be seen that as expected, the latency coding approach generates much sparser raster plots than the rate coding counterpart. It is also important to note that rate coding introduces noise in the signal as it is a probabilistic sampling which could help in the noise resilience of the training on the input data.

6.2 Network Architecture

In the following tests, the trained networks are fully connected with exponential decay synapses. As it has been seen in the previous chapter, the decay factor heavily affects the behaviour of the system. For all performed experiments, a decay factor of $\alpha = 0.5$ was chosen as it seems to offer reduced amplification of the signal while creating a good amount of memory as can be seen in the FI and hysteresis plots respectively. Furthermore, the synapse input function that prevents the neuron from saturating was chosen to range between -5 and +5 with $\rho = 5$. As we can see in the FI curve for the $\alpha = 0.5$ synapse, this range maximises the non-saturating FI range of neurons in the modulation values of interest.

Regarding the configuration of the neurons themselves, we have chosen to use the values determined in the previous chapter to obtain all-or-none spikes. The final system of equations for the synapse and neuron used in the network can be seen in 6.3.

$$\begin{cases} i[t+1] = 0.5 \cdot i[t] + W_s s_{\text{in}}[t] \\ x[t+1] = 5 \cdot \tanh\left(\dfrac{i[t+1]}{5}\right) \\ h[t+1] = \tanh\left(x[t+1] + 4 \cdot h[t] - 7 \cdot (h_{s0} + h_s[t])^2 + b_h\right) \\ h_s[t+1] = z_s(h[t]) \odot h_s[t] + (1 - z_s(h[t])) \odot h[t] \\ s_{out}[t+1] = = ReLU(h[t+1]) \\ z_s(x) = 0.9 - 0.9 \dfrac{1}{1 + \exp(-20 * (x-1))} \end{cases} \tag{6.3}$$

At a network level, we have trained the network using different modulation biases to observe the effect on the performance of firing types. We have performed these tests on a reduced-size network with 1 input

layer of 32 neurons and 1 hidden layer of 10 neurons fully connected to the output integrators.

6.3 Training Method

To perform the training we used the cross entropy loss function typically used for classification as defined for the SRC and shown in equation 6.4 for our use case with 10 classes. In this equation $\hat{y}_y$ denotes the integrator value for the correct class at the end of the sequence and $\hat{y}_c$ denotes the value for the other classes. This will result in the network pushing the charge of the correct integrator up and minimising the charge in the others.

$$l(\hat{y}, y) = -\log\left(\frac{\exp(\hat{y}_y)}{\sum_{c=1}^{10}\exp(\hat{y}_c)}\right) \tag{6.4}$$

The networks have learnable parameters in their current biases b_h, the synaptic connections W_s, and the modulation bias h_{s0}. These parameters are initialised as follows:

- Current biases are initialised to $b_h = 1$ as this is a value close to the firing threshold. It has empirically been observed that this helps learning converge faster.
- Synaptic connections W_s are initialised using the Xavier[74] uniform initialisation rule as it alleviates the vanishing and exploding gradient problems.
- The modulation bias h_{s0} has been fixed to a set firing type for several tests. When used as a learnable parameter, it has been uniformly initialised within values of $h_{s0} \in [0.5, 1.3]$ to keep a symmetry between the different firing types.

The weights are modified using the ADAM optimiser [75] and learning rates of 0.001 for both the biases and the synaptic weights. In order to prevent exploding gradients, gradient clipping is used with a maximum gradient value of 1.

During training, the $ReLU(\cdot)$ activation functions are bypassed during the backward pass to allow gradients to pass through the network even when no spikes take place. This was done by replacing the derivative of the function with the unit function defining it as $ReLU(\cdot)' = 1$.

Finally, each training has been performed on 90% of the dataset using the 10% left for validation, and over 50 epochs as it has been observed to be sufficient to attain convergence in most instances. Training has been repeated 5 times in order to assess training stability.

6.4 Fixed Modulation Rate Coding Tests

It can be seen in figure 6.2 that good overall accuracies are achieved with respect to the small size of the network which only contains 42 neurons. We see that modulation does impact performance in the rate-coded task and the best type is type2* which obtains 91% validation accuracy. The worst performing is type 2 with only 58% accuracy. These results may result surprising as intuitively type1 neurons should be the most suitable type for rate-coded tasks. Furthermore, there is little difference between the onset FI curves of type 2 and 2* but their performance varies widely.

Figure 6.2 also show encouraging results regarding the use of backpropagation on our bio-inspired modulable RNN spiking cell as training is stable for all modulation values. Indeed, we can see that the min-max area remains close to the average. Furthermore, the training converges rapidly in the first 10 epochs and progresses slowly after.

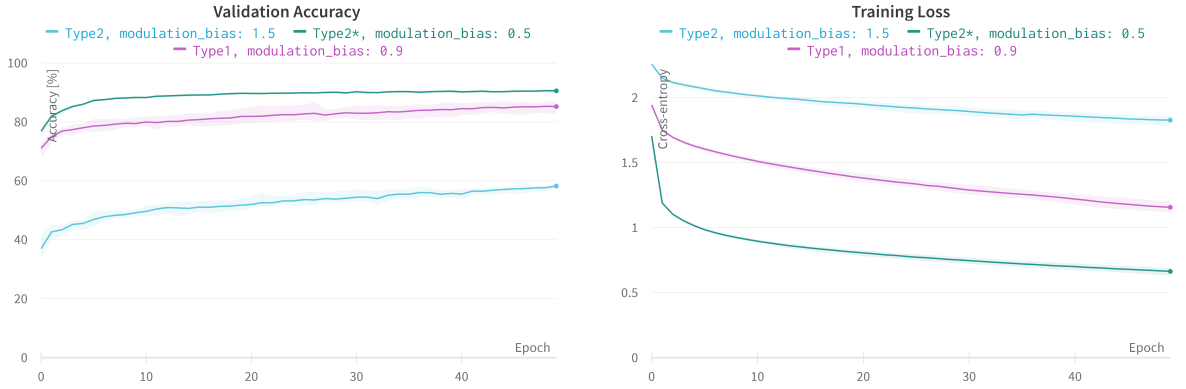


Figure 6.2: Evolution of validation accuracy and loss during training of a small mSRC network modulated in each firing type for the rate coded task. Solid lines show the mean over 5 trainings and shaded areas are bounded by min/max values of the trainings.

6.4.1 Network Dynamics Analysis

In order to better understand the performances and verify that the neurons behave as desired, in this section, we will delve deeper into the network dynamics. We will verify that spiking frequency and timing are the main carriers of information in the network. To do so, we will compare the behaviour of the networks using their average spiking time, max spike time, inter-spike intervals(ISI), spike widths, and spike heights. Spiking time is computed as percentage of time steps where the value of the fast variable is above 0 over a trace. In addition to the ISI it gives an indication of how much activation the is in the network on average with the mean spike time and the maximum recorded activation for a sample with the maximum spike time. Furthermore, samples of the dynamics through raster plots and neuron and synapse state plots can be found in the Appendices for a better view of the dynamics in a network.

From the spiking data in figure 6.3 we can see that on average type 2* neurons spike more often than the other types for a given image, followed by type 1 and then type 2. This trend is the same as what can be observed in the accuracies and could lead us to think that more spiking is necessary to carry and process more information. This observation is surprising as it would imply that the network does not effectively use the Input strength to frequency mapping to carry the task, as we have seen that type 1 yields worst accuracies than type2*. Another explanation for this result would be that the network is able to harness the hysteresis in type2* neurons and its increased FI curve after the onset. Finally, it has been observed during the development of the cell and in preliminary tests that type2* can in fact use its closeness to saturation to its advantage to modulate the spike width and use that as the main source of its computational capabilities. In order to remove that unwanted behaviour the variable $z_s(\cdot)$ function was introduced.

In order to assess the efficacy of the variable $z_s(\cdot)$ in preventing spike width modulation in the following sections we present figures where the statistics regarding spike width are compiled. The statistics were obtained through summation over the whole MNIST dataset and averaged over the 5 models with the same modulation bias.

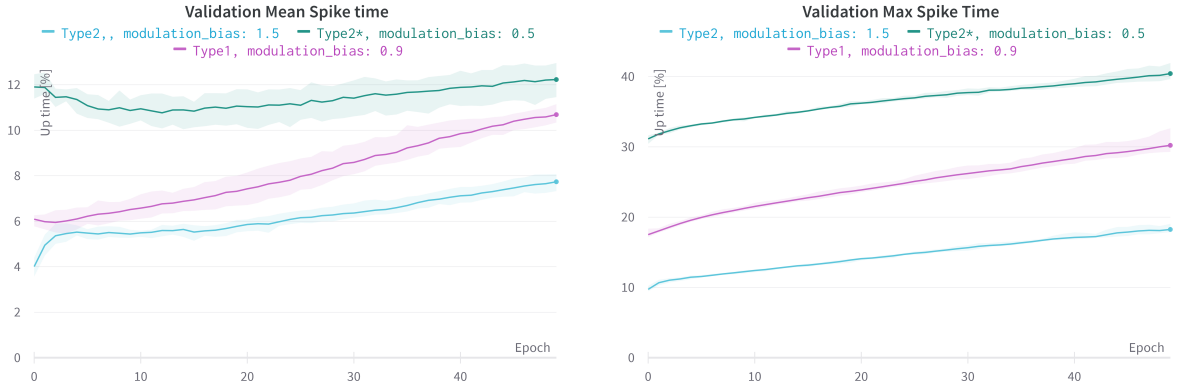


Figure 6.3: Validation mean, and max spike time as a percentage of time steps throughout the training of a small mSRC network modulated in each firing type for the rate coded task. Lines show the mean over 5 trainings and shaded areas are bounded by obtained min/max values.

a Type 2

In figure 6.8 statistics regarding the transmitted values through the network s_{out} have been compiled. Namely, the width of the spike has been computed as the number of consecutive positive values in each pulse and the height has been recorded as the amplitude of all positive values of s_{out}. From figure 6.8 we can see that the design of the cell in type 2 modulation mainly produces all-or-none spikes as most values are in the $[0.8, 1]$ range. Furthermore, with type 2 modulation, we can also see that most spikes are either composed of 2 or 3 values. These two plots indicate that the network is not able to modulate the height and the width of the spikes to transmit information in type 2.

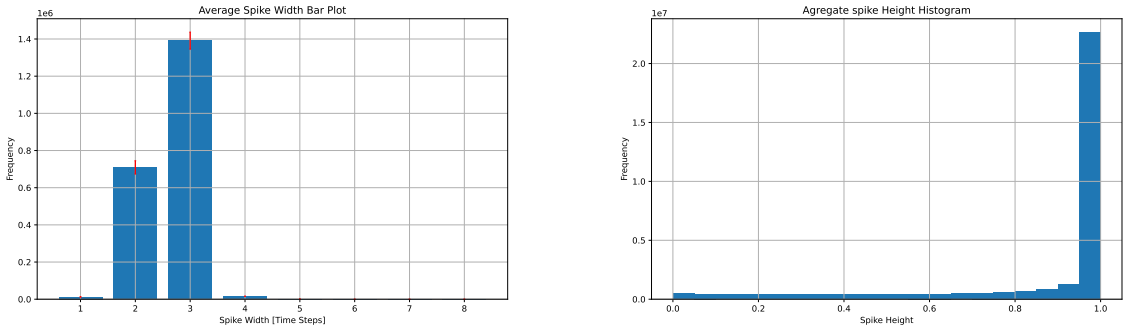


Figure 6.4: Spike width and height histograms obtained by inference over the whole rate coded MNIST dataset of a type2 mSRC network. Spike width is counted as consecutive positive values whereas height is counted as values above 0 in the voltage trace. For spike width, data is averaged over 5 trained models for spike width where the standard deviation is shown as a red bar on top of each bar. Height data frequency is accumulated over the same 5 models.

In figure 6.5 we can see the inter-spike intervals(ISI) in the traces of the Type 2 network over the whole MNIST dataset. The plot shows a steep Poisson-like distribution around a 20 time step ISI, which is an expected distribution for type2 neuron which does not have a wide firing frequency range like the two other types.

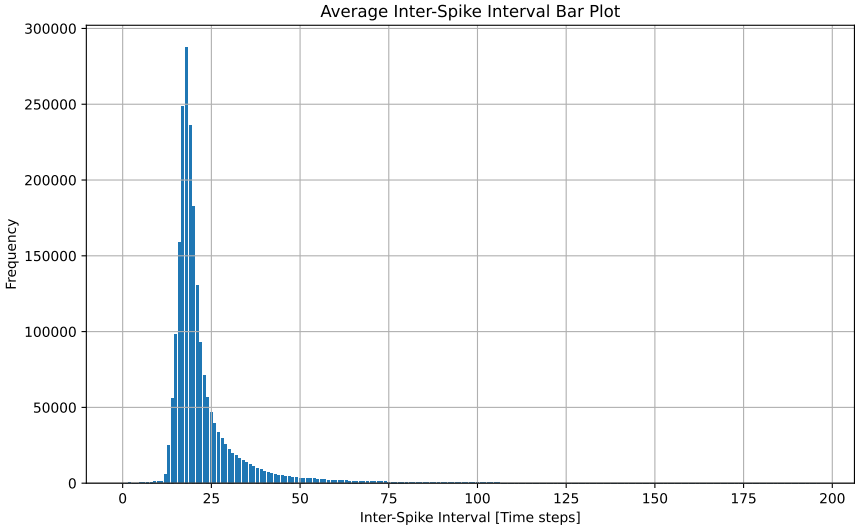


Figure 6.5: Inter-Spike Intervals (ISIs) obtained by inference over the whole rate coded MNIST dataset of a type2 mSRC network. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models.

b Type 1

Increasing the modulation bias to obtain a type 1 does not significantly change the spike widths and height as shown in figure 6.6. Most spikes are still 2 or 3 time steps wide. We can however see that the maximum width of the spikes increases to 10 but their occurrence is so sparse that they are not visible in the plot. In the previous section, it was discussed that increasing the modulation bias has the adverse of decreasing the saturation value. As the saturation moves closer this may cause a decrease in the speed of the system close to the peak value causing some spikes to be larger. Nevertheless, the occurrence of these wide spikes is negligible compared to the average spike width value, which indicates that the network is not able to leverage spike width modulation for computation.

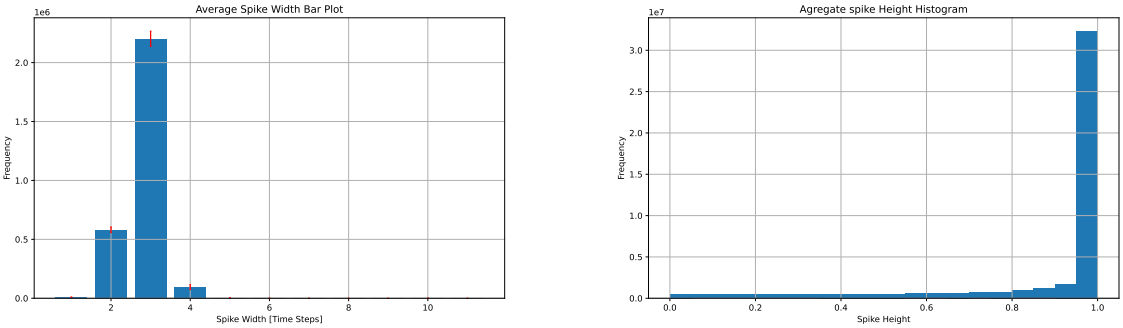


Figure 6.6: Spike width and height histograms obtained by inference over the whole rate coded MNIST dataset of a type1 mSRC network. Spike width is counted as consecutive positive values whereas height is counted as values above 0 in the voltage trace. For spike width, data is averaged over 5 trained models for spike width where the standard deviation is shown as a red bar on top of each bar. Height data frequency is accumulated over the same 5 models.

Surprisingly, figure 6.7 shows an ISI plot very similar to the one obtained with the type2 network. Type 1 having a more significant frequency range it would be expected that the plot would be flatter. However, it seems that the high activity of the network that was already observed in the mean spike time hinders the ability of the network to use the whole frequency range of type 1 neurons.

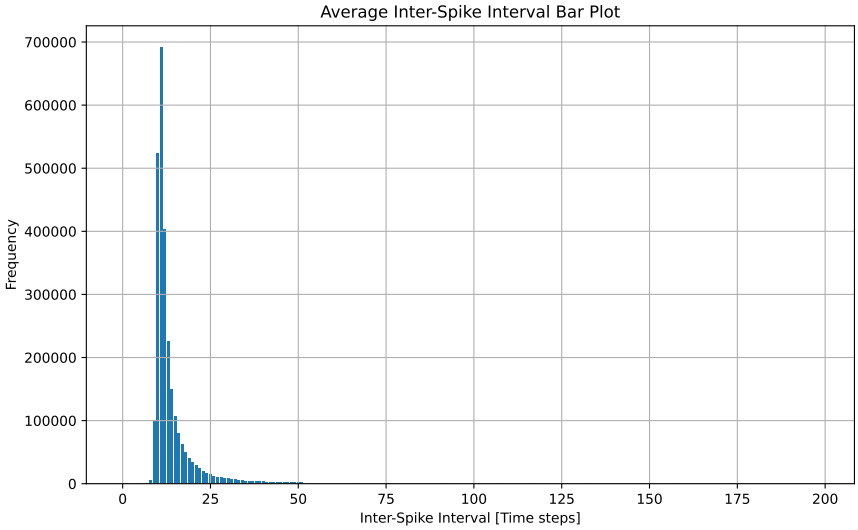


Figure 6.7: Inter-Spike Intervals (ISIs) obtained by inference over the whole rate coded MNIST dataset of a type1 mSRC network. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models.

c Type 2*

Type 2* further increases the maximum width of the spikes as shown in figure 6.8. In this modulation, the closeness of saturation seems to affect the system enough to change the modal spike width and increase it to 4. Nevertheless, the spike width and height variance remain small enough to suggest that even with wider spikes the network is not able to use it to code information.

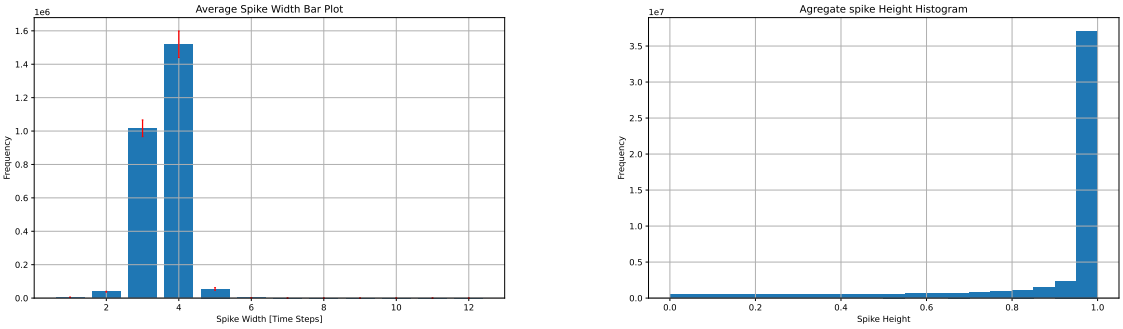


Figure 6.8: Spike width and height histograms obtained by inference over the whole rate coded MNIST dataset of a type2* mSRC network. Spike width is counted as consecutive positive values whereas height is counted as values above 0 in the voltage trace. For spike width, data is averaged over 5 trained models for spike width where the standard deviation is shown as a red bar on top of each bar. Height data frequency is accumulated over the same 5 models.

The ISI plot of type 2* in figure 6.9 shows an expected further narrowing of the used frequencies compared to type 1. When observing the behaviour of the network coupled with the performance, it is interesting to see that higher activity in the network and a seemingly narrower band of used spiking frequencies obtain higher accuracies. In this setting consisting of a small network and the rate-coded MNIST task, high average activity coupled with very distinct activity consisting of silence or high-frequency firing seems to yield the best results.

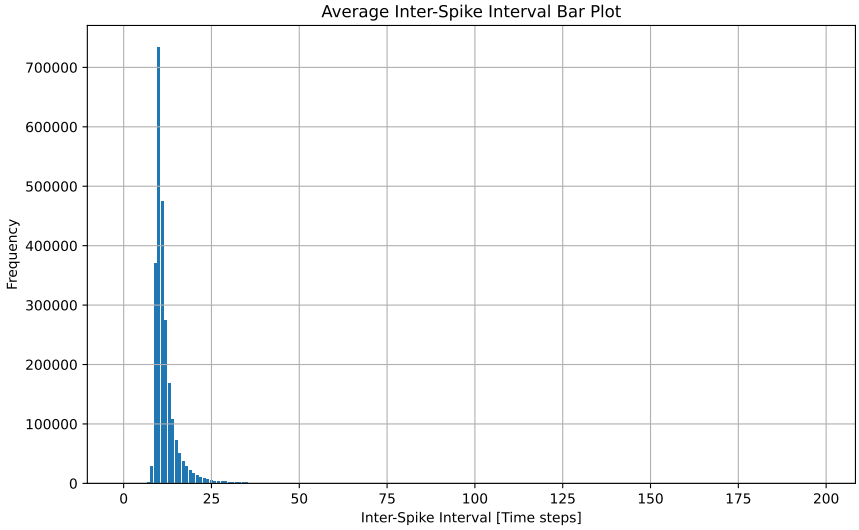


Figure 6.9: Inter-Spike Intervals (ISIs) obtained by inference over the whole rate coded MNIST dataset of a type2* mSRC network. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models.

6.5 Fixed Modulation Latency Coding Test

6.5.1 Performance

For the Latency coded task, we can see in figure 6.10 that the network's performance was poorer than in the rate task. The best-performing modulation remains Type2* but this time only achieves 73% accuracy in the best epochs. This time it was expected that Type2* would be the best-performing neuron as it is the only modulation that creates a form of memory on top of the memory present in the exponential decay synapses.

From the loss plot, we can also see that convergence is slower for the latency-coded task. Both of these results were expected as this coding of the task is much more difficult than its rate-coded counterpart as this one requires memory and a relative measure of timing between the different input spikes. Furthermore, the training is also more difficult as there is less stimulation and therefore less activity to generate the gradients that allow for convergence during the backward pass.

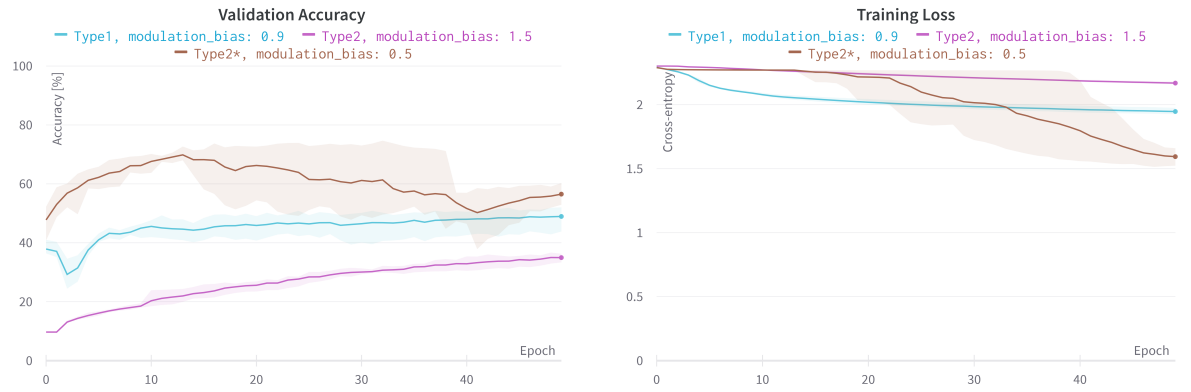


Figure 6.10: Evolution of validation accuracy and loss during training of a small mSRC network modulated in each firing type for the latency coded task. Solid lines show the mean over 5 trainings and shaded areas are bounded by min/max values of the trainings.

6.5.2 Network Analysis

One of the main goals of the latency-coded task was to have sparser activity in the network which would directly transfer to less energy expenditure in a neuromorphic hardware implementation of the network. At this, the network did succeed as we can see in figure 6.11 that both the max spike time and mean spike time for all modulation values are lower than the lowest values in the rate-coded task.

We can see that contrary to what we observe in the rate-coded task, there is a wide variation in activity between models of the same type, for type 2* networks in particular. This might be due to the hysteresis property of the firing type as high activity is required for the onset but once the onset is obtained activity remains high for longer.

Finally, we can see that in the latency-coded task performance of the network is not directly linked to its activity. Indeed, the epochs with the best accuracies are located in the 15 to 35 epoch range which corresponds to lower values of mean spike time compared to the last epochs with lower accuracies.

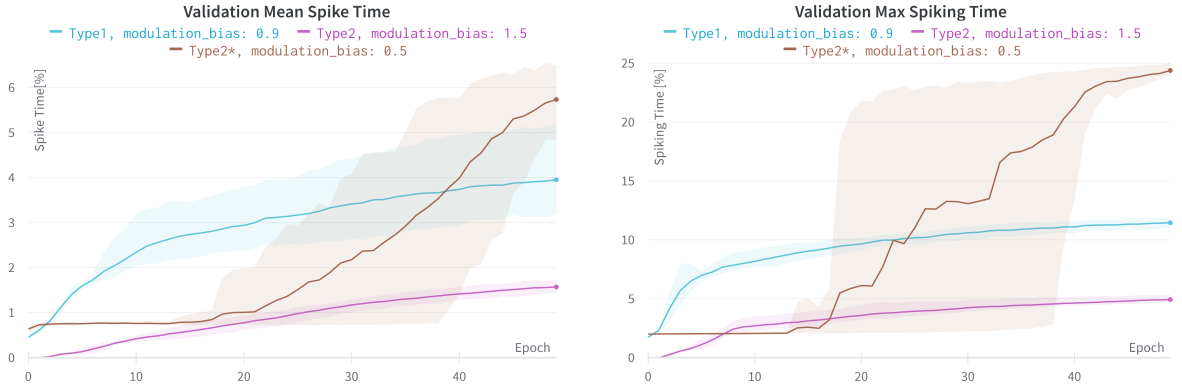


Figure 6.11: Validation mean, and max spike time as a percentage of time steps throughout the training of a small mSRC network modulated in each firing type for the latency coded task. Lines show the mean over 5 trainings and shaded areas are bounded by obtained min/max values.

Figure 6.12 shows that changing from the rate to the latency-coded task does not change the behaviour of the neurons or the ability of the network to harness spike height and width to perform computations as expected.

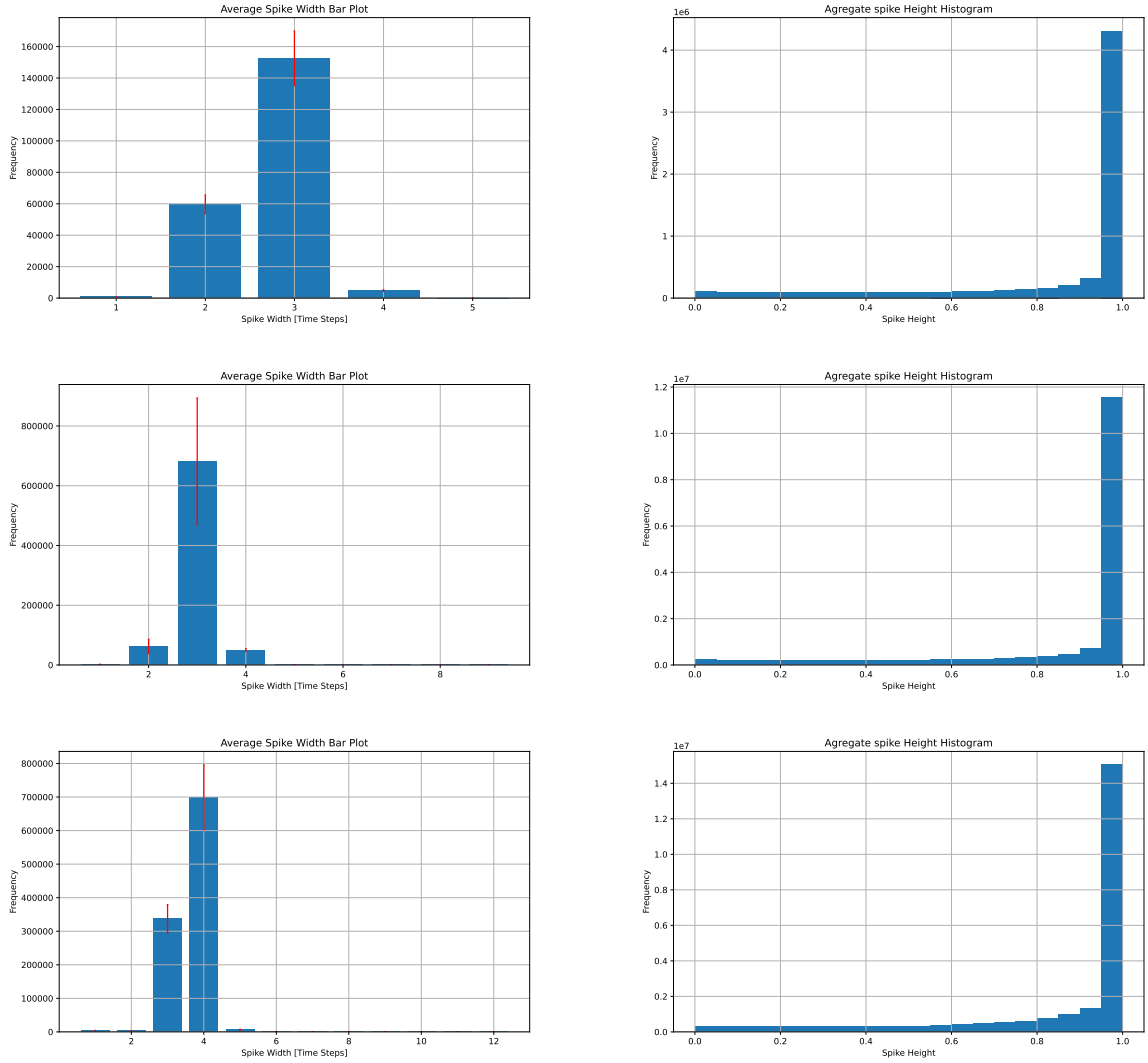


Figure 6.12: Spike width and height histograms obtained by inference over the whole latency coded MNIST dataset. From the top row, the used mSRC networks are modulated in type 2, 1, and 2* respectively. Spike width is counted as consecutive positive values whereas height is counted as values above 0 in the voltage trace. For spike width, data is averaged over 5 trained models for spike width where the standard deviation is shown as a red bar on top of each bar. Height data frequency is accumulated over the same 5 models.

a Type 2

In figure 6.13 we can see that the ISI distribution is much flatter, suggesting that latency coding exploits better the frequency range of neurons. This is also due to the exponential decay synapses that probably excite neurons in a wider range of currents.

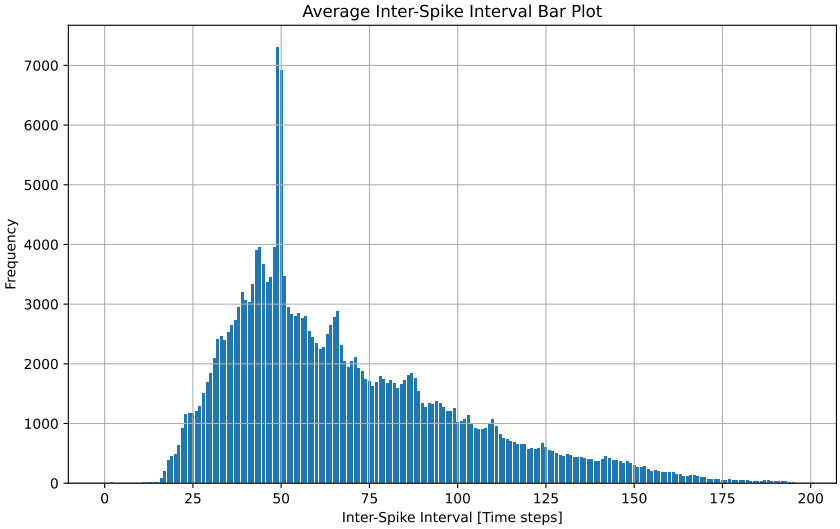


Figure 6.13: Inter-Spike Intervals (ISIs) obtained by inference over the whole latency coded MNIST dataset of a type2 mSRC network. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models.

b Type 1

In figure 6.14 we again see a flatter distribution of the ISI distribution is observed due to the sparser latency coding of the task. We also see that the overall activity of the type 1 network is higher than type 2 which is expected as the threshold is lower so the network can spike for longer during the synapse discharge. Furthermore, we also see a wider peak of the distribution which is expected because of the larger frequency range of type 1 neurons.

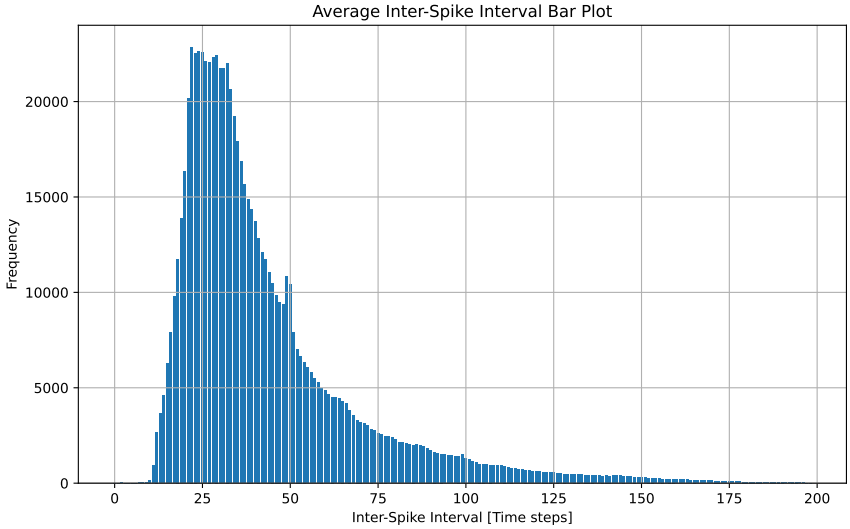


Figure 6.14: Inter-Spike Intervals (ISIs) obtained by inference over the whole latency coded MNIST dataset of a type1 mSRC network. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models.

c Type 2*

The higher activity of the type 2* network that has been seen in the mean spike times plots is again seen in figure 6.15 where the maximum frequency is 20 times higher than in type 2 and 7 times higher than in type 1. This shows the effect of hysteresis as the network needs to push excitation high in order to obtain a first response and then the decay of the synapse coupled to a higher frequency range after the spiking onset will obtain longer-lasting activity.

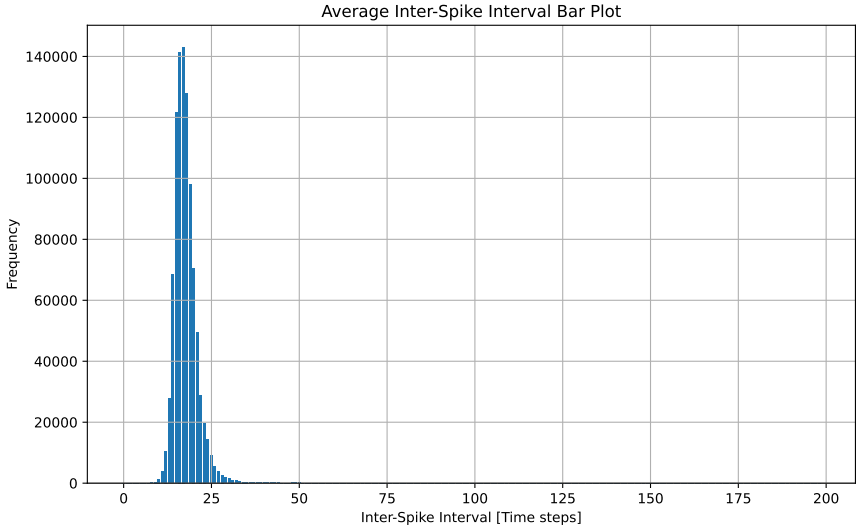


Figure 6.15: Inter-Spike Intervals (ISIs) obtained by inference over the whole latency coded MNIST dataset of a type2* mSRC network. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models.

6.6 Training with modulation as a parameter

In the following experiments, the modulation bias h_{s0} was set as a learnable parameter of the network and its value was uniformly initialised. With this test, we can determine the importance of heterogeneity in the performance of this task. Furthermore, in these tests, we observe how the network modifies the values of the modulation parameter to optimise the performance on the task.

6.6.1 Rate-Coded Task

a Performance

In figure 6.16 we can see that setting the modulation as a learnable parameter does not hinder learning as we obtain 91% accuracy consistently. We can also see that the loss decreases rapidly within the first 10 time steps as it does with fixed modulation which further shows that modulation can be learned effectively with backpropagation.

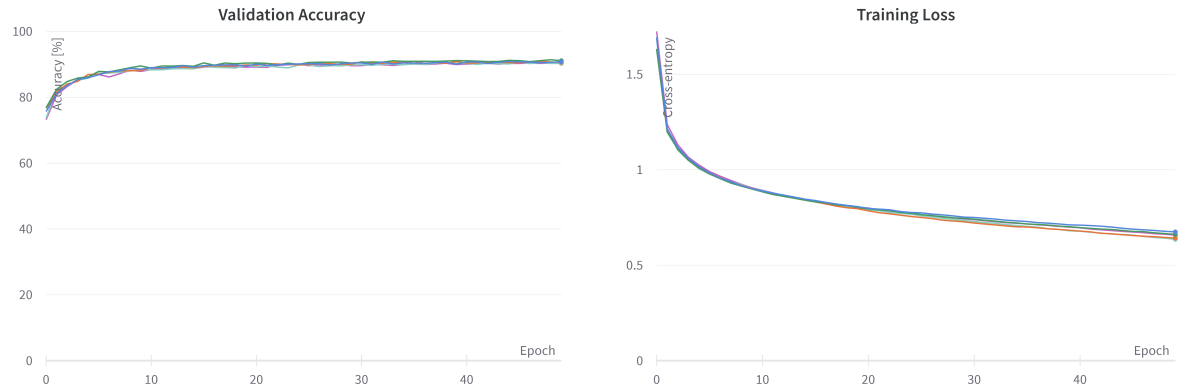


Figure 6.16: Evolution of validation accuracy and loss during training of 5 small mSRC networks with modulation as a learnable parameter on the rate coded MNIST.

b Network Analysis

The data regarding spike time in figure 6.17 shows activity throughout training that is comparable to fixed modulation in type 2*. When looking at the modulation biases through training in figure 6.18 we can see that the network converges towards type 2* values. This confirms that in a small network for the rate-coded task, the behaviour of learnable modulation converges toward type2* behaviour. This could signify that type 2* firing has the most computational power and the more useful properties for the current task. Furthermore, the learning of this task suggests that there is no advantage in having a heterogeneous population of neurons or that backpropagation is not able to harness heterogeneity effectively to complete the task.

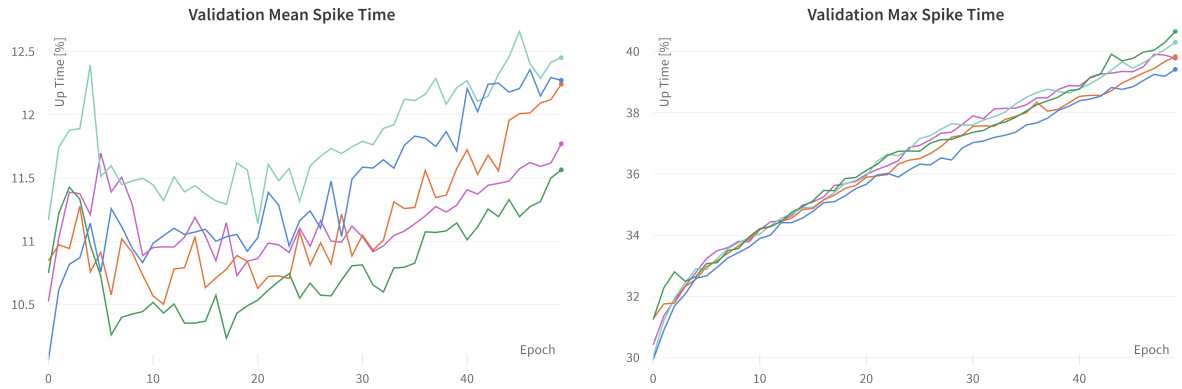


Figure 6.17: Validation mean, and max spike time as a percentage of time steps throughout the training of 5 small mSRC networks with the modulation as a learnable parameter for the rate coded task.

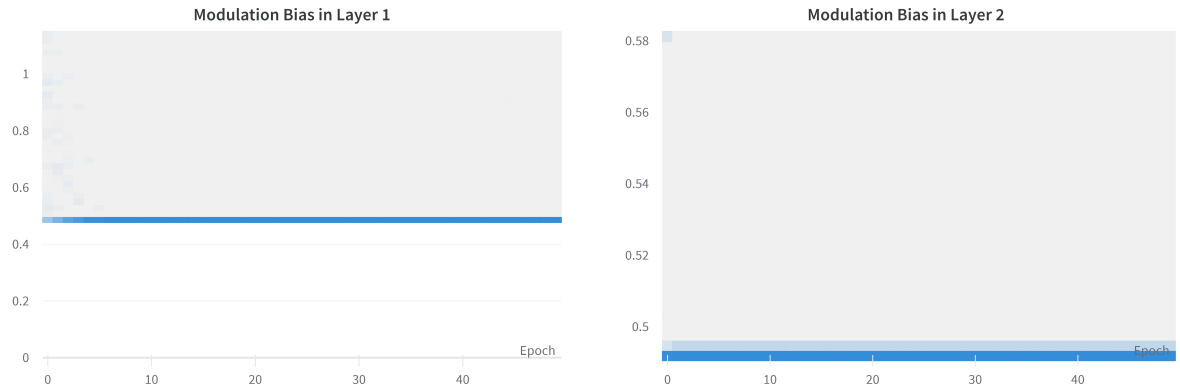


Figure 6.18: Evolution of the modulation value h_{s0} throughout the training of an mSRC network with the modulation as a learnable parameter for the rate coded task.

6.6.2 Latency-Coded Task

a Performance

Again, figure 6.19 shows that learning the modulation obtains a similar performance to fixed modulation in the latency-coded MNIST. However, we can see that the convergence is faster and has a lower variance suggesting that smoothly varying the modulation parameter helps with learning in this task.

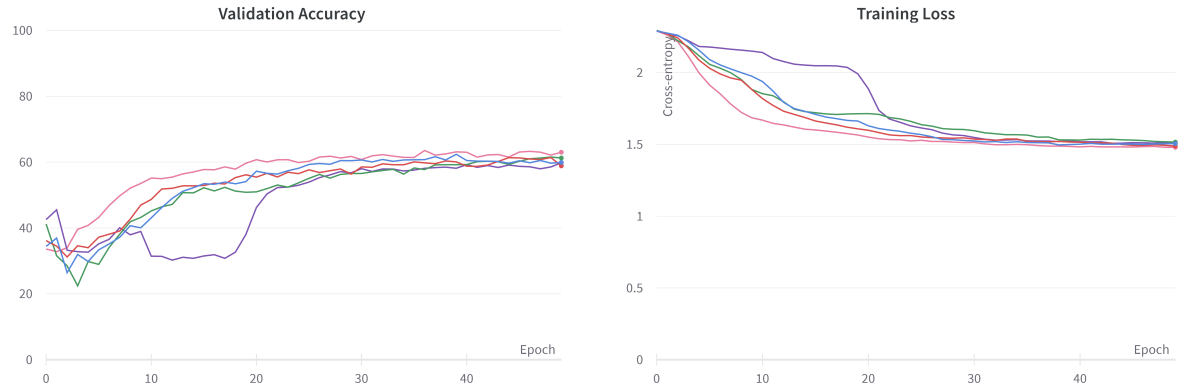


Figure 6.19: Evolution of validation accuracy and loss during training of 5 small mSRC networks with modulation as a learnable parameter on the latency coded MNIST.

b Network Analysis

In figure 6.20 we can see that making modulation learnable yields a network that has higher average activation than fixing the parameter. In the plots of figure 6.21 we can see that for the latency-coded MNIST, modulation also tends to decrease to enter type 2* firing.

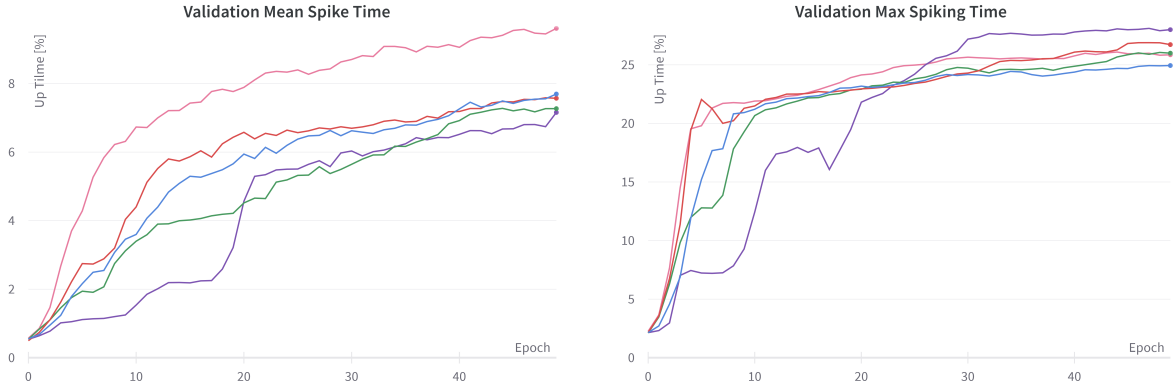


Figure 6.20: Validation mean, and max spike time as a percentage of time steps throughout the training of 5 small mSRC networks with the modulation as a learnable parameter for the latency coded task.

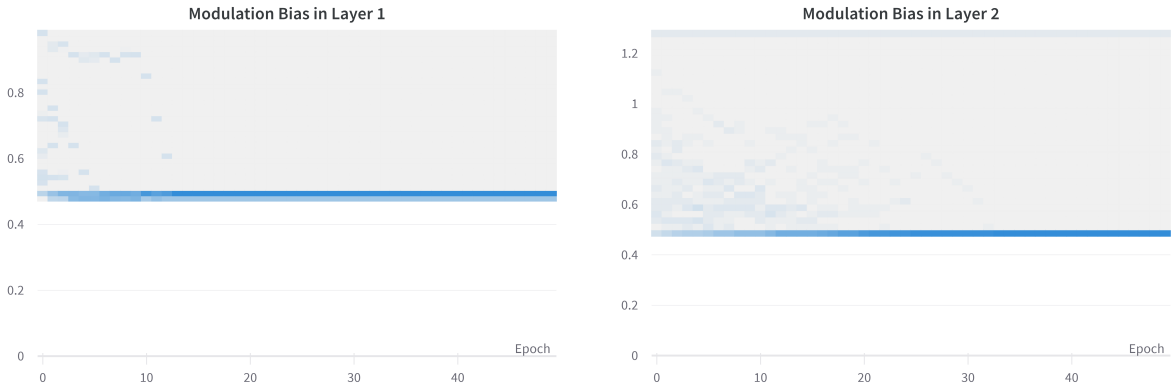


Figure 6.21: Evolution of the modulation value h_{s0} throughout the training of an mSRC networks with the modulation as a learnable parameter for the latency coded task.

6.7 Training Robustness in Spite of Aliasing

As discussed in the previous chapter, the discretisation of a continuous time system to create a discrete-time RNN cell will cause the high frequencies to be aliased. In order to study the effect of aliasing on the performance and spiking behaviour of the system we modified the $z_s(\cdot)$ function by decreasing the z_s^{hyp} value from $z_s^{hyp} = 0.9$ as it is in the previous test to $z_s^{hyp} = 0.5$. This has the effect of highly accelerating the slow feedback effect, effectively reducing the spike amplitude in continuous time and causing aliasing when discretised. Figure 6.22 shows the continuous time system using the new $z_s^{hyp} = 0.5$ value and the trace in discrete time generate with an RNN cell using the same z_s^{hyp} value. As it can be seen, spikes that do not have full amplitude only reaching 0.5 in the continuous case are all-or-none in the discretised case due to high-frequency behaviour being aliased.

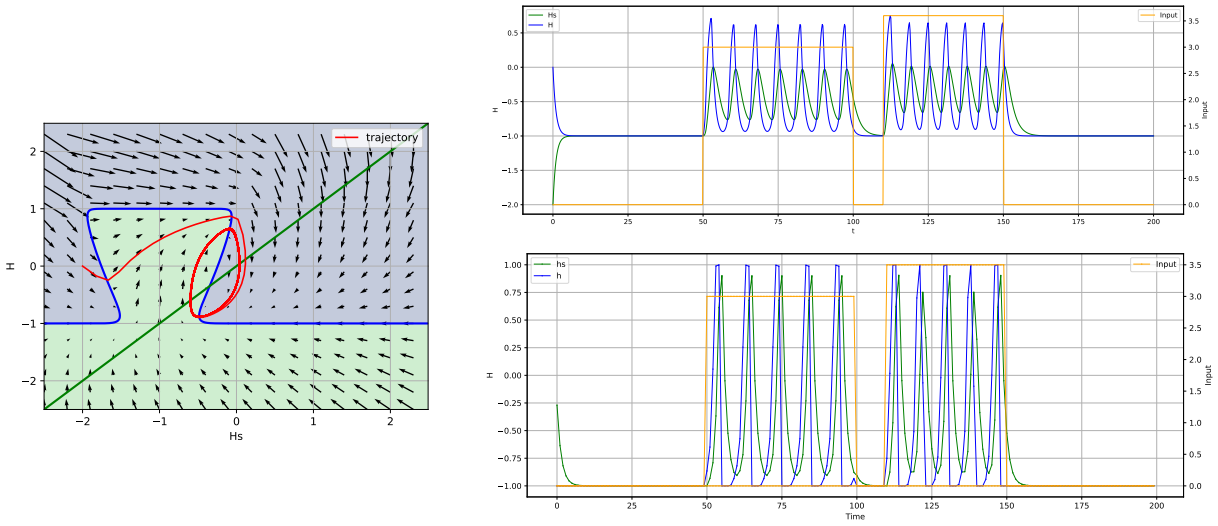


Figure 6.22: Phase portrait with a trajectory of the system with $z_s^{hyp} = 0.5$ on the left. On the right, the traces of the system without aliasing on top. The bottom trace shows the simulation with an RNN cell that introduces discretisation noise and aliasing.

Figure 6.23 shows that aliasing does not hinder learning. Increasing the speed of z_s by setting $z_s^{hyp} = 0.5$ actually improves accuracy in the rate coded task. In the latency coded task the no value of z_s^{hyp} has a clear advantage. However, it is important to note that high aliasing is a source of error that would not be present in a neuromorphic hardware implementation of the network. Therefore, it is unknown if the results obtained in simulation where high aliasing is present would translate well to real application performance.

Despite faster spikes, in figure 6.24 we can see that the networks with lower z_s^{hyp} have higher up time than their slower z_s counterpart. This shows that increasing the speed of z_s increases the frequencies that can be observed in the network. The increase in activity does correlate with accuracy as it has previously been observed. However, in this case the activity doubles in some cases but accuracy is only marginally better when comparing $z_s^{hyp} = 0.5$ to $z_s^{hyp} = 0.9$

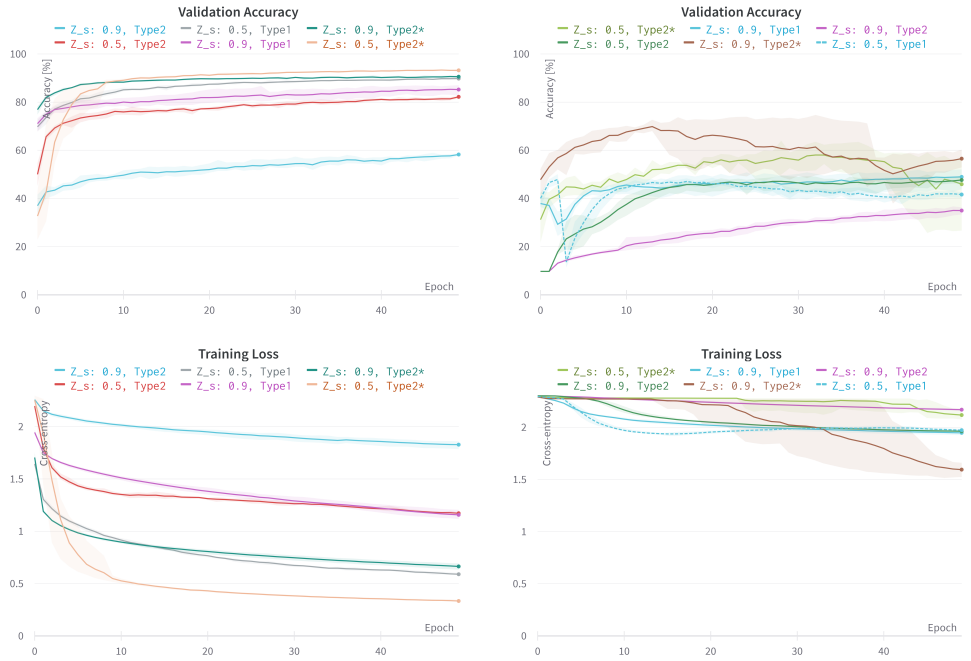


Figure 6.23: Validation accuracies and training loss for all the trained modulations and values of z_s^{hyp}. Plots in the left column corresponds to results of the rate coded task, latency coded is on the right.

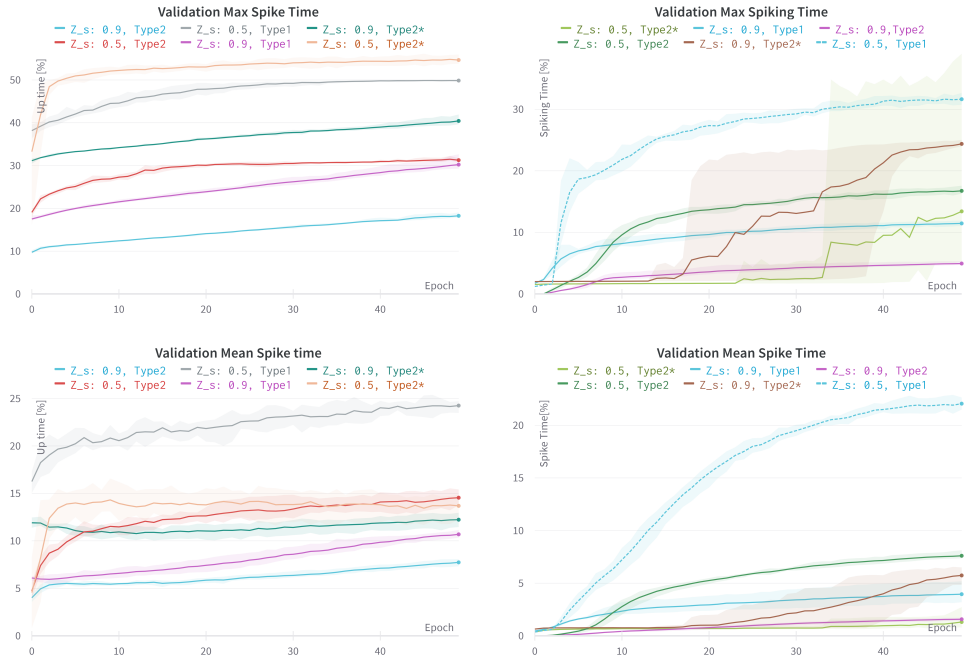


Figure 6.24: Validation max and mean spike time for all the trained modulations and values of z_s^{hyp}. Plots in the left column corresponds to results of the rate coded task, latency coded is on the right.

Statistics regarding the spike widths heights and inter-spike intervals were also computed for the $z_s^{hyp} = 0.5$ networks. For the sake conciseness these plots have been placed in appendices C.1 and C.2 for the rate

coded task and C.3 and C.4 for the latency coded task. In those plots we see that the faster dynamics make the spikes shorter in general but slightly increase the variance of the spike heights. The increase in different spike heights does not seem to be significant enough for the network to use it as a means of computation. Regarding the ISIs, we observe the same behaviours than with $z_s^{hyp} = 0.9$ but with ranges that start lower and that are more concentrated towards higher frequencies. This further indicates that increasing the speed of the slow dynamics of the system dramatically increases the activation of the network.

Chapter 7

Conclusion and Further Work

In this thesis, we have started by reviewing the state-of-the-art techniques and approaches to neuronal excitability modelling. We have discussed the main branches of Machine Learning along with their shortcomings and opportunities, going more into detail regarding Recurrent Neural Networks. We have seen how these networks, much like neurons, are feedback systems in nature. In particular, the recently proposed Spiking Recurrent Cell(SRC) harnesses this feedback nature to create a cell that is able to simulate spiking while learning through backpropagation. To analyse the behaviour of this cell, we have used the Finite Difference Method to transform the SRC into a continuous time system that could then be analysed using classical non-linear systems tools. The analysis of the SRC cell has shown that it is only capable of type2-like excitability which is only one of the 3 main types of excitability discussed at the beginning of this thesis. Furthermore, the analysis showed that the SRC does not make all-or-none spikes which is an essential aspect of neurons which code information in either their firing frequency or timing. Seeking to solve this problem we have proposed 2 recurrent cells based on the SRC. The first consists of a modified version baptised the modulable-SRC(mSRC) that finds inspiration in state-of-the-art reduced models of excitability. The mSRC is able to smoothly switch between type 2, 1, and 2* excitability through the help of a single additional parameter. Furthermore, in this cell, the slow dynamics of the equation have been modified to obtain spikes that are closer to the all-or-none biological spikes. The second cell consists of an extension of the mSRC that adds a feedback equation with an ultra-slow timescale that allows for another fundamental excitability behaviour that is bursting. Finally, these cells designed in continuous time have been translated back into their RNN cell equivalent through the same Finite Difference Method. It has been noted, that even though designing recurrent cells in continuous time and translating them into implementable RNNs is a very convenient mode of operation, the Finite Difference Method does have shortcomings and care should be taken when using it. In particular, one should keep in mind that not all discretisations are stable and some can have exploding numerical errors. Moreover, due to sampling errors high frequencies might become distorted due to aliasing if the time step is chosen too large for a given cell.

Having designed the mSRC cell, we have tested it and verified its correct behaviour with satisfactory results. With only 42 fully connected neurons arranged in 2 layers accuracies of 90% were obtained on the rate coded MNIST dataset. This result, even if it is below the state of the art for classical machine learning, is very encouraging taking into account the small size of the network and the fact that these results should be compared to spiking neural networks and not classical ML. Furthermore, one of the main goals of this project was to design a cell that could learn its modulation through backpropagation which we succeeded at doing. Surprisingly, the network converges to a purely type2* network to obtain the best results in both the rate-coded and latency-coded tasks.

Further tests to assess the network learning resistance towards high levels of aliasing were also carried out. From these tests, we have seen that aliasing and a change in the slow feedback speed have little impact

on learning and even obtained higher results in some settings compared to the lower aliasing counterpart. Nevertheless, it must be noted that if the end goal of the network is to be implemented in neuromorphic hardware, aliasing during training will make the performance on hardware unpredictable and probably worse as aliasing will not be present in hardware.

Many results in this thesis are encouraging and make a good case for the proof of concept of a modulable spiking recurrent cell. However, this is a very new approach to SNNs and the cell has shortcomings that ought to be addressed. Modulation increases the variance in spike width which is an undesired effect of the model. Furthermore, the use of the squared function in a machine learning application has bad properties for learning it squashes values that are smaller than one in absolute terms and amplify the rest. A maybe more suitable function that would yield the effect of symmetry would be the absolute value function, however, tests should be done to verify that it does improve training performance without compromising on the functionality of the cell. Regarding the obtained results when modulation is a learned parameter, the lack of use of heterogeneity by the network could be due to the reduced scope of the task or the small size of the network. In order to validate this idea tests with a larger network should be carried out on more sophisticated tasks. Such tasks could be more closely related to neuroscience tasks or reinforcement learning as they often involve long-term memory dependencies and require the network to make different outputs over time. Furthermore, these tests used the cross entropy loss with integrators at the output which forced the activity to be as high as possible for the correct output. Although effective this loss function may not be the best one can produce for a sparse spike generation.

Regarding the network architecture, the fully connected model is extremely flexible but is also difficult to train. One of the next steps that could be taken to improve the performance of RNN-based spiking neural networks would be to use other architectures like conventional architectures for image recognition. Furthermore, the addition of synapses gives a new dimension to neural network design as synapses could be connected together to form synaptic trees as they have been observed in biological neurons. The choice of synapse model is also a crucial one that has not been explored in detail in this thesis but we have seen how much they can affect the dynamics of the network.

It is also important to remember that the end use of SNNs is to create low-power intelligent systems. Implementing these networks in hardware is also an avenue that should be researched as it will give further insights into how to develop models that are more readily hardware implementable and that conserve their performance in spite of all the challenges that hardware would offer like noise, components mismatch, and chip footprint to name a few. As part of another project linked to this one, the mSRC neurons were successfully implemented on a field programmable array(FPGA). FPGAs offer high levels of parallelism which is useful regarding inference speed, however, they are still digital in nature meaning that there is no advantage regarding power efficiency. One notable challenge in that implementation was the use of nonlinear functions which are very resource intensive in a setting where resources are very limited. Simplifying even further the design of the mSRC system could not only benefit its implementation on an FPGA but also allow for faster inference on classical computers resulting in a training speed up.

It is needless to say that this novel SNN approach is still in its infancy and requires more work in order to improve it and test it further. Nevertheless, the preliminary results are very encouraging, and the ease and flexibility of implementation offered by using well-established frameworks such as PyTorch will allow rapid iterations to come to life. It is exciting to see the growing interest in neuroscience-inspired machine learning. Undoubtedly, it will bring progress to both fields and help shed light on one of the most fundamental mysteries of nature which is learning.

Bibliography

[1] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[2] Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. Gpt-4 passes the bar exam. *SSRN*, 2023.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[4] Kovác L. The 20 w sleep-walkers. *EMBO Rep*, 2010.

[5] Yuniesky Andrade-Talavera, André Fisahn, and Antonio Rodríguez-Moreno. Timing to be precise? an overview of spike timing-dependent plasticity, brain rhythmicity, and glial cells interplay within neuronal circuits. *Molecular Psychiatry*, Mar 2023.

[6] L.F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5):303–304, 1999.

[7] A L Hodgkin and A F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol*, 117(4):500–544, August 1952.

[8] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking neural networks and their applications: A review. *Brain Sci*, 12(7), June 2022.

[9] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[10] Guillaume Drion. *Regulation of Excitability, Pacemaking, and Bursting: Insights from Dopamine Neuron Electrophysiology*. PhD thesis, ULiège - Université de Liège, 25 February 2013.

[11] Membrane potential membrane potential is generated by the differential distribution of ions. 2014.

[12] R Llinás, M Sugimori, and S M Simon. Transmission by presynaptic spike-like depolarization in the squid giant synapse. *Proceedings of the National Academy of Sciences*, 79(7):2415–2419, April 1982.

[13] J A Connor, D Walter, and R McKown. Neural repetitive firing: modifications of the Hodgkin-Huxley axon suggested by experimental results from crustacean axons. *Biophys J*, 18(1):81–102, April 1977.

[14] Guillaume Drion, Timothy O'Leary, and Eve Marder. Ion channel degeneracy enables robust and tunable neuronal firing rates. *Proceedings of the National Academy of Sciences*, 112(38), September 2015.

[15] Guillaume Drion, Timothy O'Leary, Julie Dethier, Alessio Franci, and Rodolphe Sepulchre. Neuronal behaviors: A control perspective. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 1923–1944, 2015.

[16] R. Sepulchre, G. Drion, and A. Franci. Control across scales by positive and negative feedback. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):89–113, 2019.

[17] Jeff Moehlis. Canards for a reduction of the hodgkin-huxley equations. *Journal of Mathematical Biology*, 52(2):141–153, Feb 2006.

[18] T B Kepler, L F Abbott, and E Marder. Reduction of conductance-based neuron models. *Biol Cybern*, 66(5):381–387, 1992.

[19] Guillaume Drion, Alessio Franci, Julie Dethier, and Rodolphe Sepulchre. Dynamic input conductances shape neuronal spiking. *eNeuro*, 2(1), 2015.

[20] R Fitzhugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophys J*, 1(6):445–466, July 1961.

[21] Alessio Franci, Guillaume Drion, and Rodolphe Sepulchre. An organizing center in a planar model of neuronal excitability. *SIAM Journal on Applied Dynamical Systems*, 11(4):1698–1722, 2012.

[22] Alessio Franci, Guillaume Drion, Vincent Seutin, and Rodolphe Sepulchre. A balance equation determines a switch in neuronal excitability. *PLoS Comput Biol*, 9(5):e1003040, May 2013.

[23] Alessio Franci, Guillaume Drion, and Rodolphe Sepulchre. Modeling the modulation of neuronal bursting: A singularity theory approach. *SIAM Journal on Applied Dynamical Systems*, 13(2):798–829, 2014.

[24] A L Hodgkin. The local electric changes associated with repetitive action in a non-medullated axon. *J Physiol*, 107(2):165–181, March 1948.

[25] David A McCormick and Michael P Nusbaum. Editorial overview: neuromodulation: tuning the properties of neurons, networks and behavior. *Curr Opin Neurobiol*, 29:iv–vii, November 2014.

[26] Steven A. Prescott, Yves De Koninck, and Terrence J. Sejnowski. Biophysical basis for three distinct dynamical mechanisms of action potential initiation. *PLOS Computational Biology*, 4(10):1–18, 10 2008.

[27] R Fitzhugh. Thresholds and plateaus in the Hodgkin-Huxley nerve equations. *J Gen Physiol*, 43(5):867–896, May 1960.

[28] John Rinzel and Bard Ermentrout. Analysis of neural excitability and oscillations. *Methods of Neuronal Modeling*, 01 1998.

[29] E Izhikevich. Dynamical systems in neuroscience. *MIT Press*, page 111, July 2007.

[30] Thomas Kepler, L. Abbott, and Eve Marder. Order reduction for dynamical systems describing the behavior of complex neurons. In R.P. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3. Morgan-Kaufmann, 1990.

[31] Alessio Franci, Guillaume Drion, Vincent Seutin, and Rodolphe Sepulchre. A novel phase portrait to understand neuronal excitability, 2011.

[32] Kathleen Jacquerie and Guillaume Drion. Robust switches in thalamic network activity require a timescale separation between sodium and t-type calcium channel activations. *PLOS Computational Biology*, 17(5):1–30, 05 2021.

[33] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[34] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989.

[35] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. When and why are deep networks better than shallow ones? *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), February 2017.

[36] Bernard Widrow, Marcian E Hoff, et al. Adaptive switching circuits. In *IRE WESCON convention record*, volume 4, pages 96–104. New York, 1960.

[37] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[38] Lorenzo Ciampiconi, Adam Elwood, Marco Leonardi, Ashraf Mohamed, and Alessandro Rozza. A survey and taxonomy of loss functions in machine learning, 2023.

[39] Akash Kumar Bhoi, Pradeep Kumar Mallick, Chuan-Ming Liu, and Valentina E. Balas, editors. *Bio-inspired Neurocomputing*. Springer Singapore, 2021.

[40] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks, 2019.

[41] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, May 2015.

[42] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[43] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[44] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[45] Li-Ye Niu, Ying Wei, Wen-Bo Liu, Jun-Yu Long, and Tian-hao Xue. Research progress of spiking neural network in image classification: a review. *Applied Intelligence*, Mar 2023.

[46] Tao Liu, Zihao Liu, Fuhong Lin, Yier Jin, Gang Quan, and Wujie Wen. Mt-spike: A multilayer time-based spiking neuromorphic architecture with temporal error backpropagation, 2018.

[47] Jibin Wu, Yansong Chua, Malu Zhang, Qu Yang, Guoqi Li, and Haizhou Li. Deep spiking neural network with spike count based learning rule. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2019.

[48] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3227–3235, 2018.

[49] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14, 2020.

[50] Haowen Fang, Amar Shrestha, Ziyi Zhao, and Qinru Qiu. Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2799–2806. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[51] Shibo Zhou, Xiaohua LI, Ying Chen, Sanjeev T. Chandrasekaran, and Arindam Sanyal. Temporal-coded deep spiking neural network with easy training and robust performance, 2021.

[52] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J. Thorpe, and Timothée Masquelier. Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern Recognition*, 94:87–95, oct 2019.

[53] Peter Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9, 2015.

[54] Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 International Joint Conference on Neural Networks, IJCNN 2017 - Proceedings*, Proceedings of the International Joint Conference on Neural Networks, pages 1999–2006. Institute of Electrical and Electronics Engineers Inc., June 2017. Funding Information: This work is partially supported by the National Science Foundation under Grants CCF-1337300. Publisher Copyright: © 2017 IEEE.; 2017 International Joint Conference on Neural Networks, IJCNN 2017 ; Conference date: 14-05-2017 Through 19-05-2017.

[55] Amirhossein Tavanaei, Zachary Kirby, and A. Maida. Training spiking convnets by stdp and gradient descent. *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018.

[56] Amirhossein Tavanaei and Anthony S. Maida. Multi-layer unsupervised learning in a spiking convolutional neural network. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2023–2030, 2017.

[57] Pierre Falez, Pierre Tirilly, Ioan Marius Bilasco, Philippe Devienne, and Pierre Boulet. Multi-layered spiking neural network with target timestamp threshold adaptation and stdp, 2019.

[58] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[59] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Front Neurosci*, 12:774, October 2018.

[60] Ruizhi Chen, Hong Ma, Shaolin Xie, Peng Guo, Pin Li, and Donglin Wang. Fast and efficient deep sparse multi-strength spiking neural networks with dynamic pruning. pages 1–8, 07 2018.

[61] Jingling Li, Weitai Hu, Ye Yuan, Hong Huo, and Tao Fang. Bio-inspired deep spiking neural network for image classification. pages 294–304, 10 2017.

[62] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[63] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[64] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[65] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[66] David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Comput*, 25(3):626–649, December 2012.

[67] Nicolas Vecoven, Damien Ernst, and Guillaume Drion. A bio-inspired bistable recurrent cell allows for long-lasting memory. *PLOS ONE*, 16(6):e0252676, jun 2021.

[68] Eve Marder, L. F. Abbott, Gina G. Turrigiano, Zheng Liu, and Jorge Golowasch. Memory from the dynamics of intrinsic membrane currents. *Proceedings of the National Academy of Sciences*, 93(24):13481–13486, 1996.

[69] Florent De Geeter, Damien Ernst, and Guillaume Drion. Spike-based computation using classical recurrent neural networks, 2023.

[70] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[71] Bard Ermentrout. Linearization of F-I Curves by Adaptation. *Neural Computation*, 10(7):1721–1729, 10 1998.

[72] C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[73] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

[74] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[75] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Appendix A

Additional Biophysical Neuron Models

A.1 Hodgkin and Huxley Model

The following equations and figure define and show simulations of the Hodgkin and Huxley Model. The first developed conductance based model, rewarded with the Nobel Prize for Physiology or Medicine in 1963.

$$\begin{cases} C_m \dfrac{dV_m}{dt} = I_{ext} - \bar{g}_K n^4 (V_m - V_K) - \bar{g}_{Na} m^3 h (V_m - V_{Na}) - \bar{g}_L (V_m - V_L) \\ \dfrac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m) n \\ \dfrac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m) m \\ \dfrac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m) h \end{cases} \tag{A.1}$$

with

$$\begin{aligned} \alpha_n(V_m) &= \frac{0.01(V_m+55)}{1-\exp(-0.1(V_m+55))} \\ \beta_n(V_m) &= 0.125\exp(-0.0125(V_m+65)) \\ \alpha_m(V_m) &= \frac{0.1(V_m+40)}{1-\exp(-0.1(V_m+40))} \\ \beta_m(V_m) &= 4\exp(-0.0556(V_m+65)) \\ \alpha_h(V_m) &= 0.07\exp(-0.05(V_m+65)) \\ \beta_h(V_m) &= \frac{1}{1+\exp(-0.1(V_m+35))} \end{aligned} \tag{A.2}$$

A.2 Connor Stevens model

The Connor Stevens model expands the HH model with additional ion channels. These additional ion channels require differential equations to model the gating mechanism as it can be seen in the equations

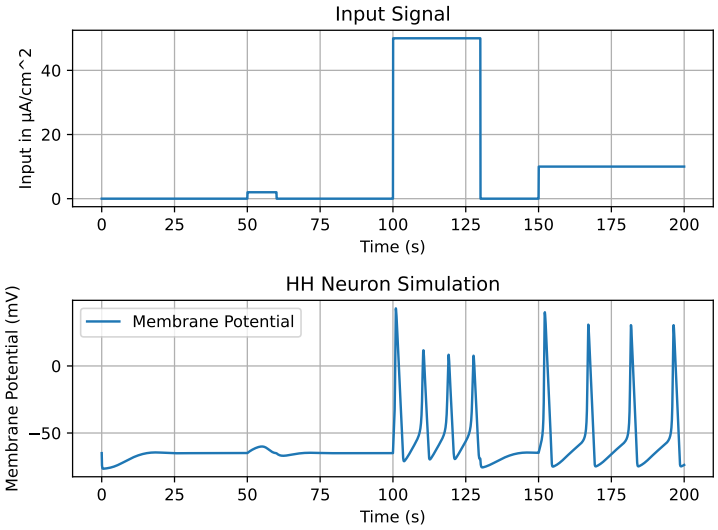


Figure A.1: Simulation over 200 time steps with a 10ms dt of a HH neuron with different input intensities and lengths.

here bellow. The model is able to recreate more firing patter but this is at the cost of increased complexity.

$$
\begin{cases}
C_m \dfrac{dV_m}{dt} = g_l(V_m - V_l) + g_{Na} m^3 h (V_m - V_{Na}) + g_K n^4 (V_m - V_K) + g_A a^3 b (V_m - V_K) + g_{Ca} m_{Ca}^2 (V_m - V_{Ca}) + I_{ext} \\
\dfrac{dm}{dt} = \dfrac{m_\infty(V_m) - m}{\tau_m(V_m)} \\
\dfrac{dh}{dt} = \dfrac{h_\infty(V_m) - h}{\tau_h(V_m)} \\
\dfrac{dn}{dt} = \dfrac{n_\infty(V_m) - n}{\tau_n(V_m)} \\
\dfrac{da}{dt} = \dfrac{a_\infty(V_m) - a}{\tau_a(V_m)} \\
\dfrac{db}{dt} = \dfrac{b_\infty(V_m) - b}{\tau_b(V_m)} \\
\dfrac{dm_{\mathrm{Ca}}}{dt} = \dfrac{m_{\mathrm{Ca}\infty}(V_m) - m\mathrm{Ca}}{\tau_{m_{\mathrm{Ca}}}}
\end{cases}
\tag{A.3}
$$

with

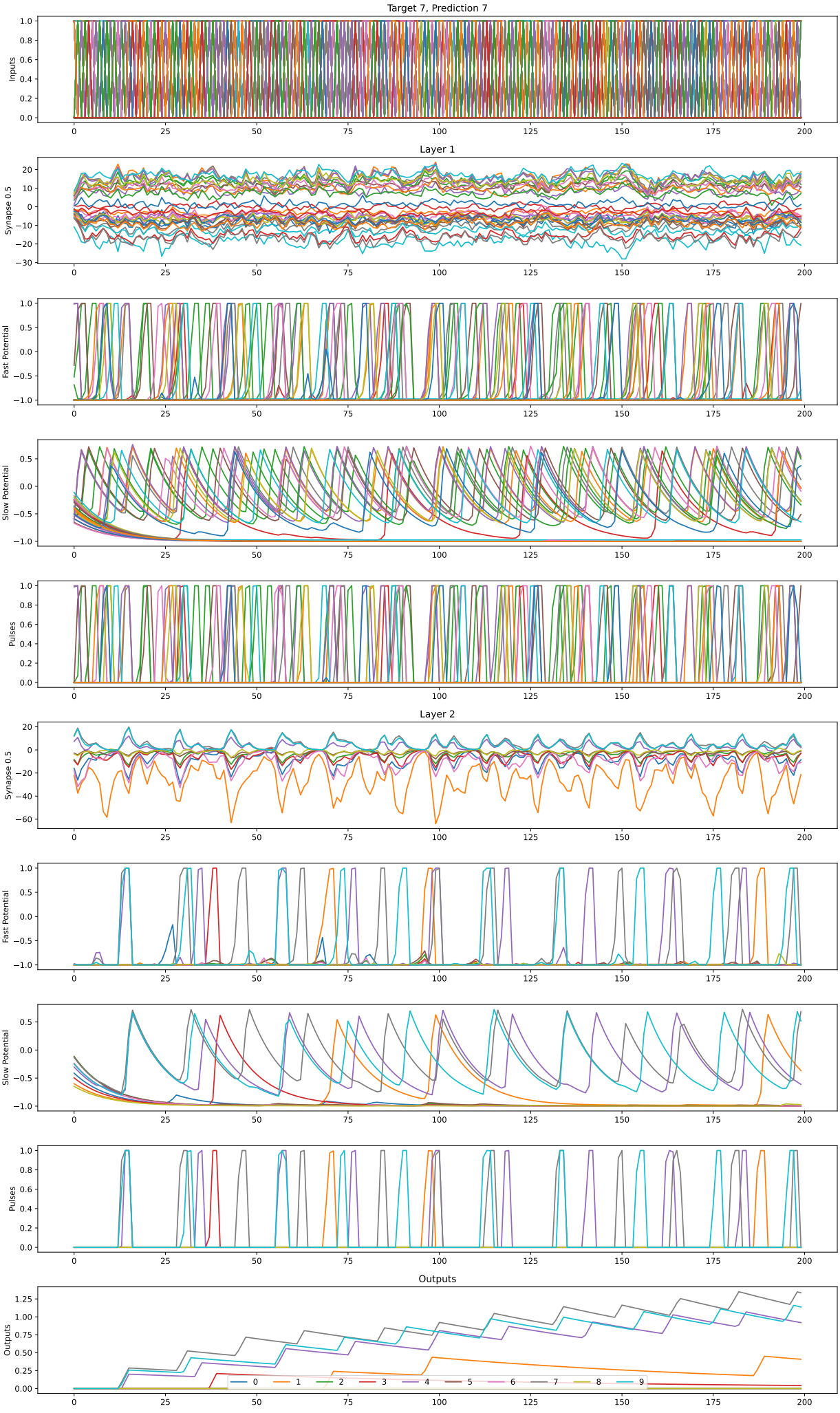
$$
\begin{aligned}
\alpha_m(V) &= 0.38\frac{V+29.7}{1-\exp(-0.1(V+29.7))} \\
\beta_m(V) &= 15.2\exp(-0.0556(V+54.7)) \\
m_\infty(V) &= \frac{\alpha_m(V)}{\alpha_m(V)+\beta_m(V)} \\
\tau_m(V) &= \frac{1}{\alpha_m(V)+\beta_m(V)} \\
\alpha_h(V) &= 0.266\exp(-0.05(V+48)) \\
\beta_h(V) &= \frac{3.8}{1+\exp(-0.1(V+18))} \\
h_\infty(V) &= \frac{\alpha_h(V)}{\alpha_h(V)+\beta_h(V)} \\
\tau_h(V) &= \frac{1}{\alpha_h(V)+\beta_h(V)} \\
\alpha_n(V) &= \frac{0.02(V+45.7)}{1-\exp(-0.1(V+45.7))} \\
\beta_n(V) &= 0.25\exp(-0.0125(V+55.7)) \\
n_\infty(V) &= \frac{\alpha_n(V)}{\alpha_n(V)+\beta_n(V)} \\
\tau_n(V) &= \frac{1}{\alpha_n(V)+\beta_n(V)} \\
\alpha_a(V) &= \left(0.0761\frac{\exp((V+94.22)/31.84)}{1+\exp((V+1.17)/28.93)}\right)^{\frac{1}{3}} \\
\tau_a(V) &= 0.3632+\frac{1.158}{1+\exp((V+55.96)/20.12)} \\
\beta_a(V) &= \left(\frac{1}{1+\exp((V+53.3)/14.54)}\right)^4 \\
\tau_b(V) &= 1.24+\frac{2.678}{1+\exp((V+50)/16.027)}
\end{aligned}
\tag{A.4}
$$

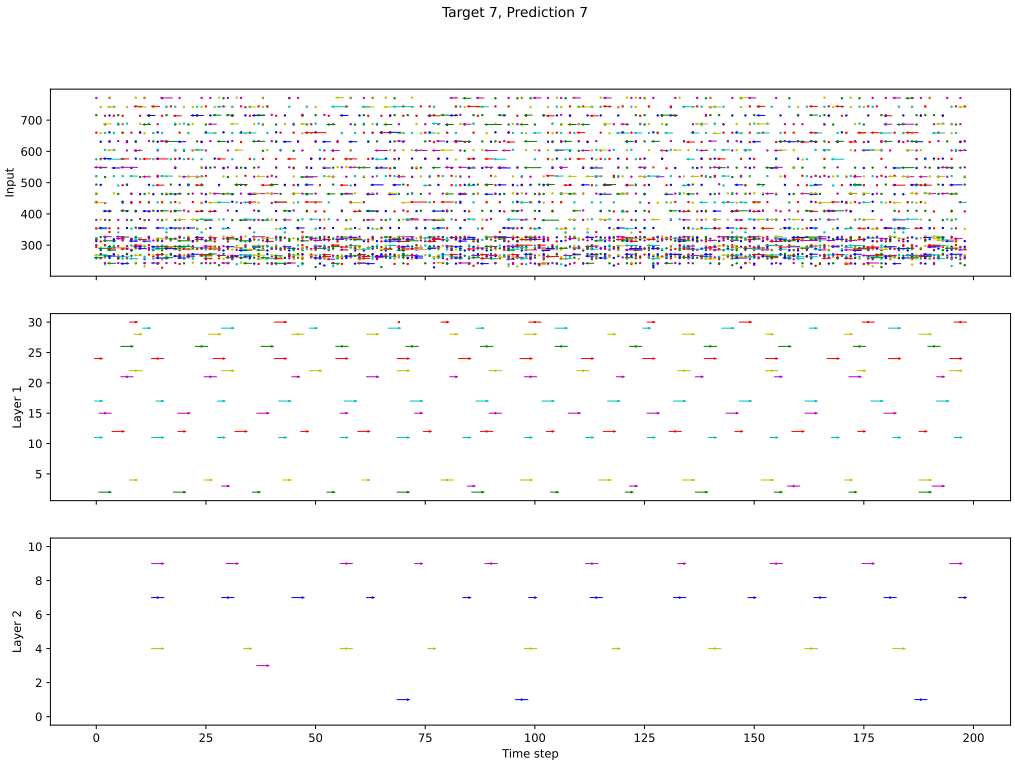
Appendix B

Additional mSRC network plots

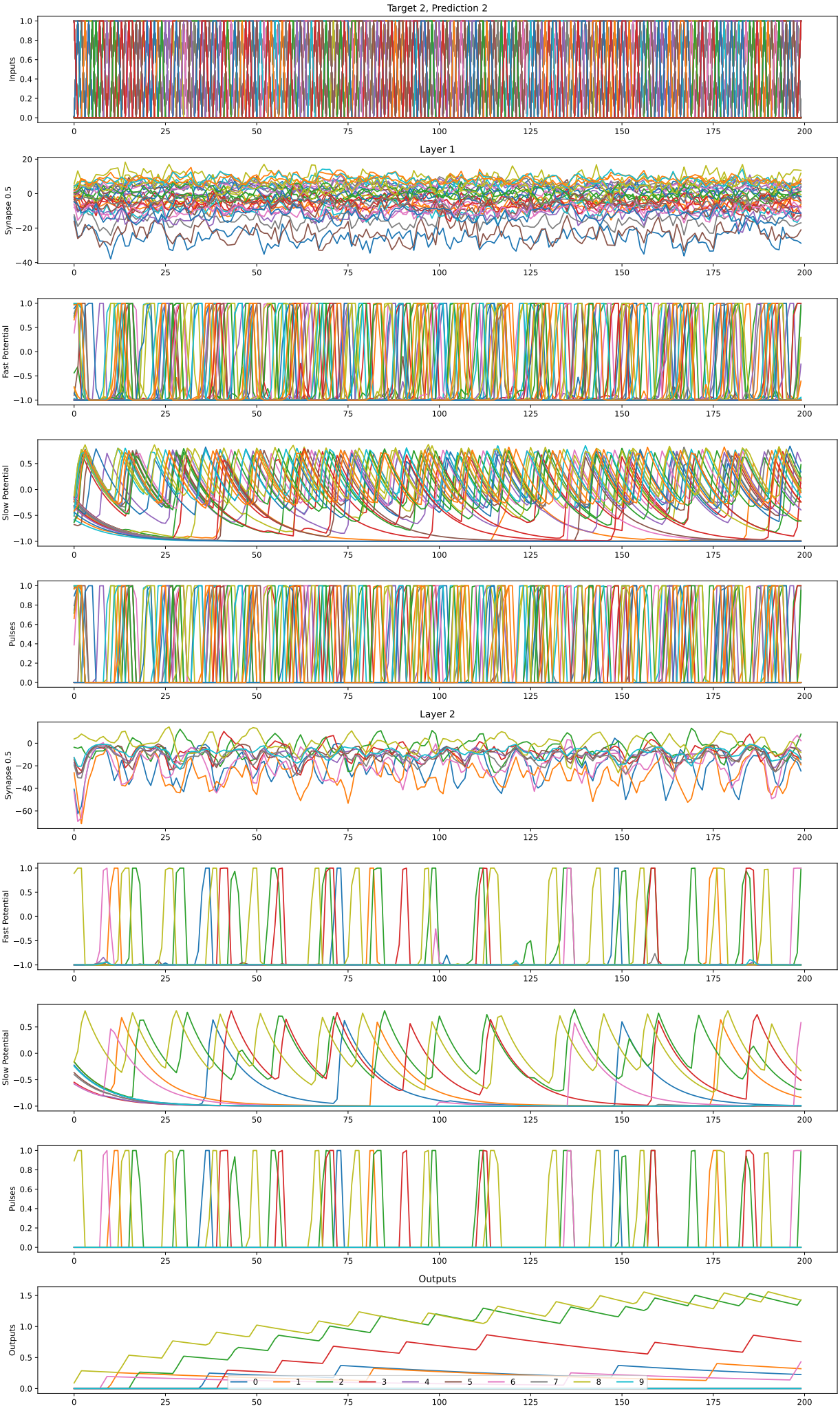
In the following section we will give detailed plots corresponding to trained mSRC networks to help visualise the behaviour of the whole network. In these plots we can see the inputs as well as the synaptic currents and cell state at each time step. The information is separated by layers and there is one pane per element of the network. Namely, for each layer the synaptic currents are given followed by the fast and slow variables of the neuron the pulses that are transmitted to the next layer. Following the state plots in each of the 3 trained modulation values, a raster plot is given summarising the pulses at the input and at each layer of the network.

B.1 Type 2 Rate Coding

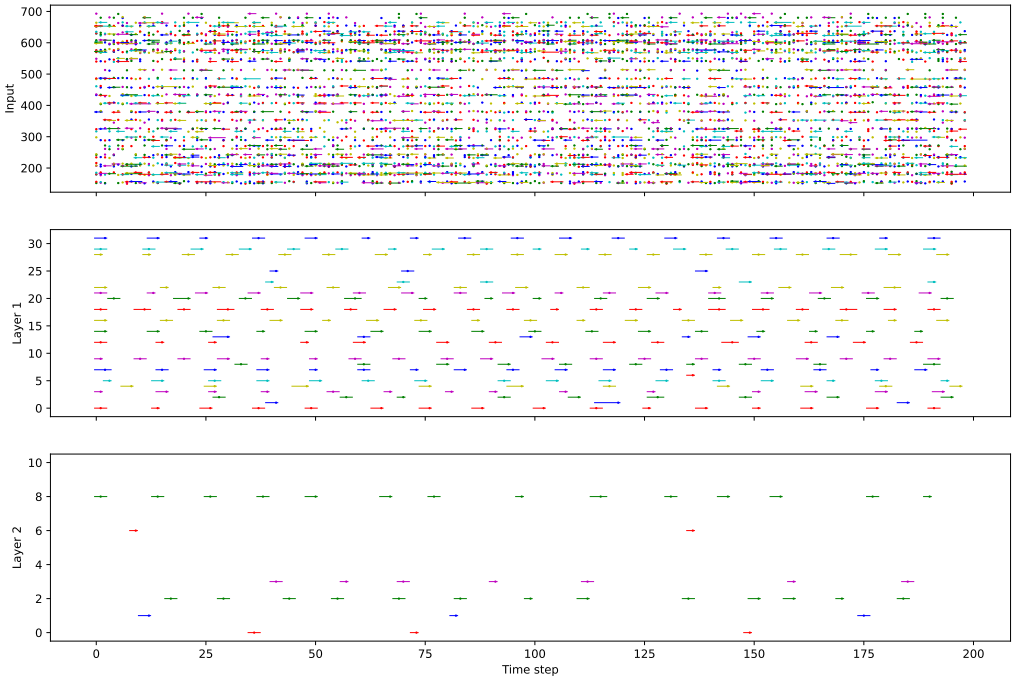

Target 7, Prediction 7
Inputs
Layer 1
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Layer 2
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Outputs
Outputs
0
1
2
3
4
5
6
7
8
9


Target 7, Prediction 7
Input
Layer 1
Layer 2
Time step

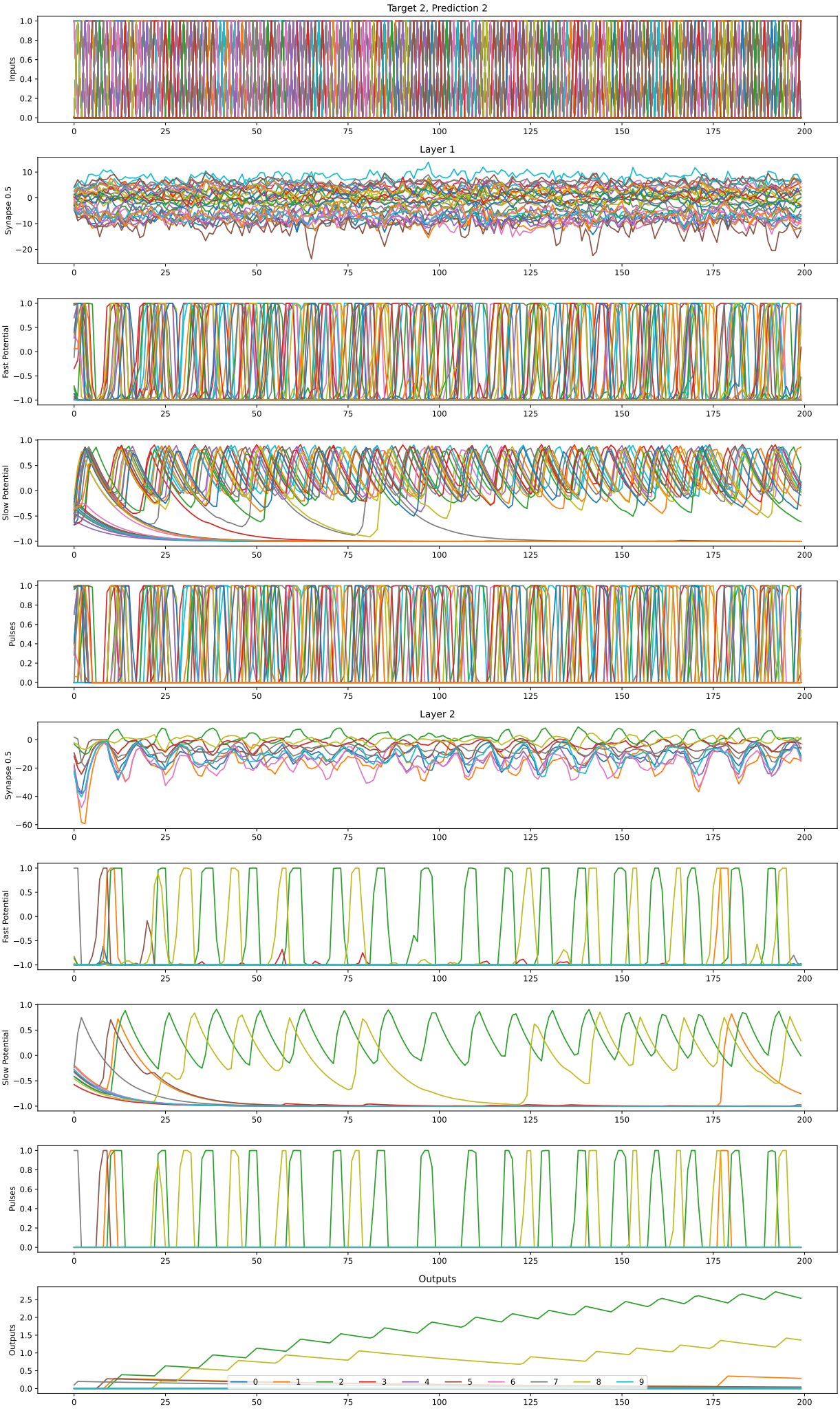
B.2 Type 1 rate coding


Target 2, Prediction 2
Inputs
Layer 1
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Layer 2
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Outputs
Outputs
0 1 2 3 4 5 6 7 8 9

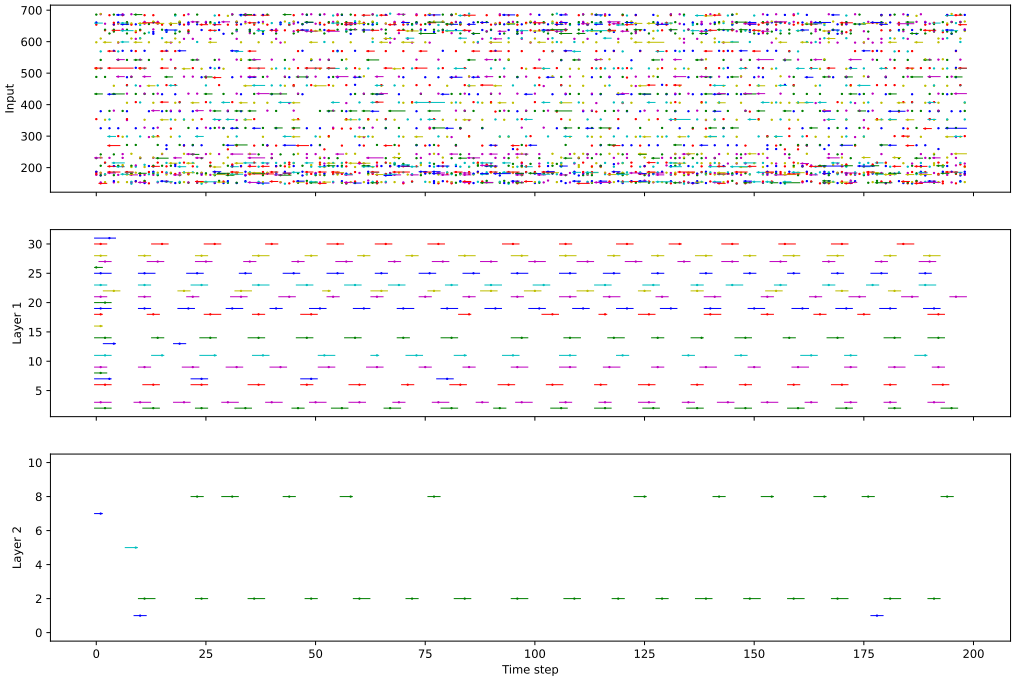
Target 2, Prediction 2



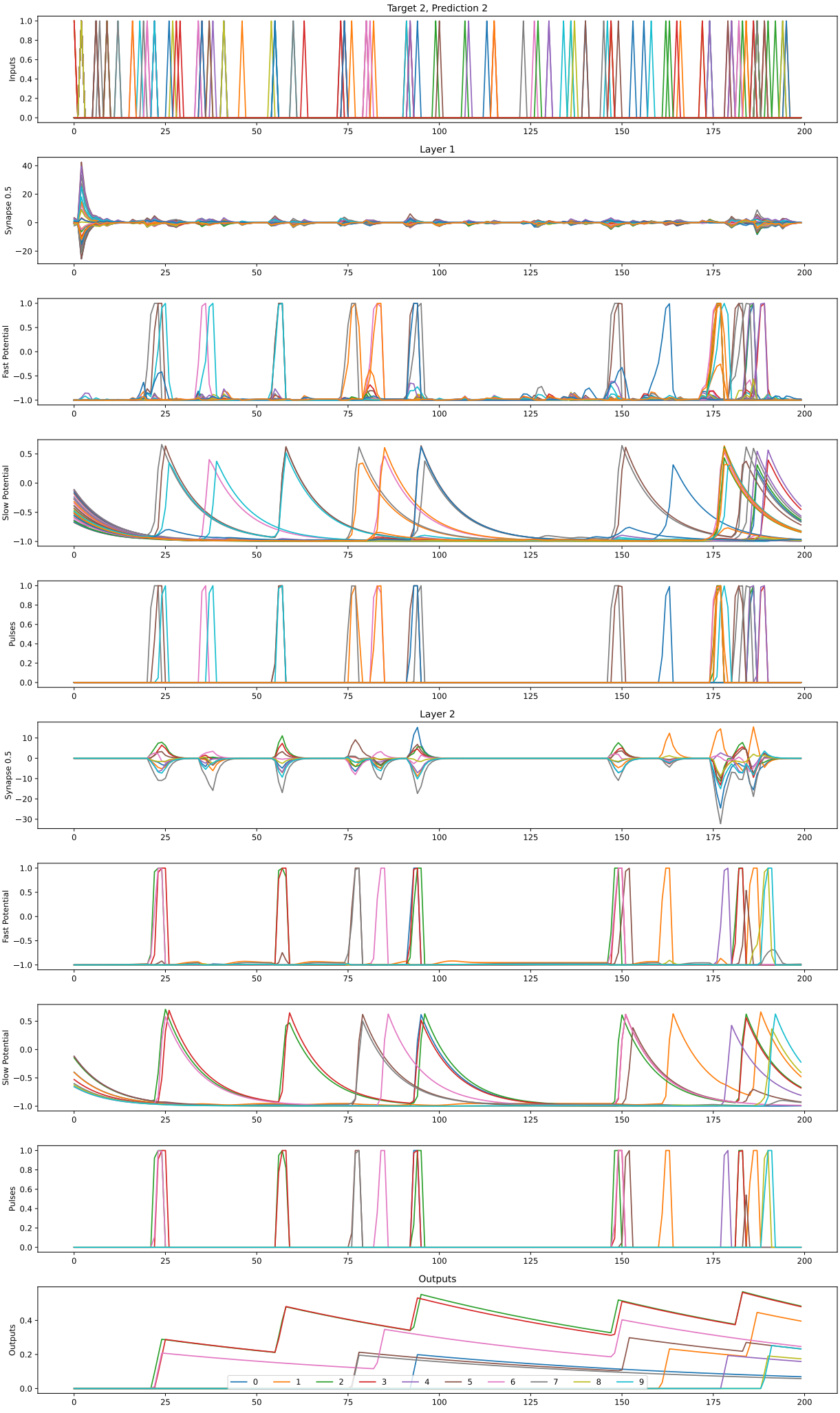
B.3 Type 2* rate coding


Target 2, Prediction 2
Inputs
Layer 1
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Layer 2
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Outputs
Outputs
0
1
2
3
4
5
6
7
8
9

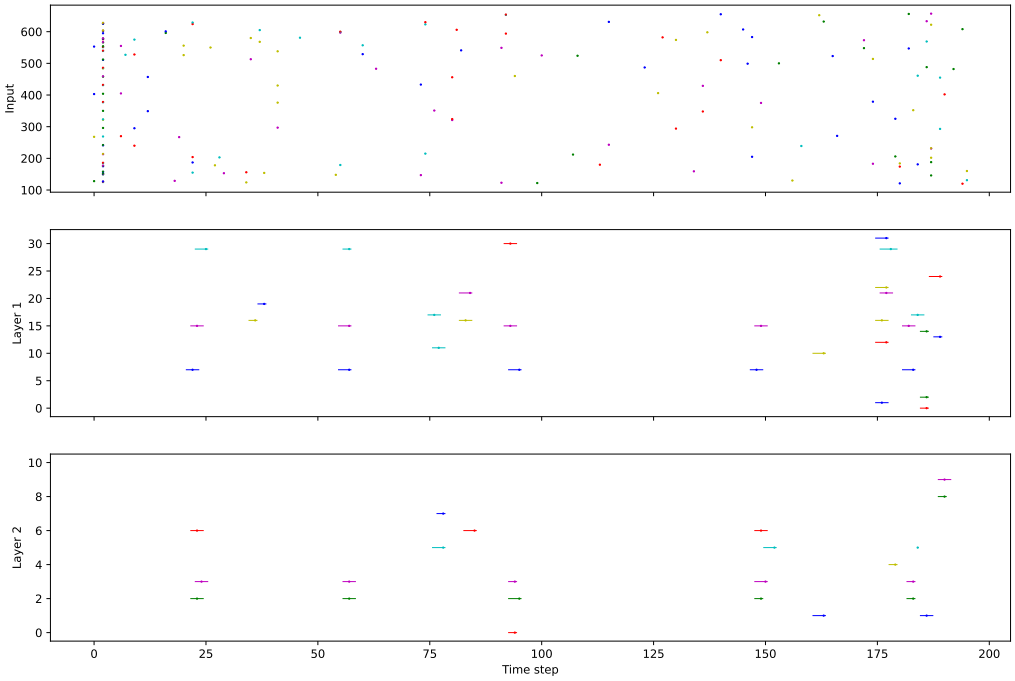
Target 2, Prediction 2



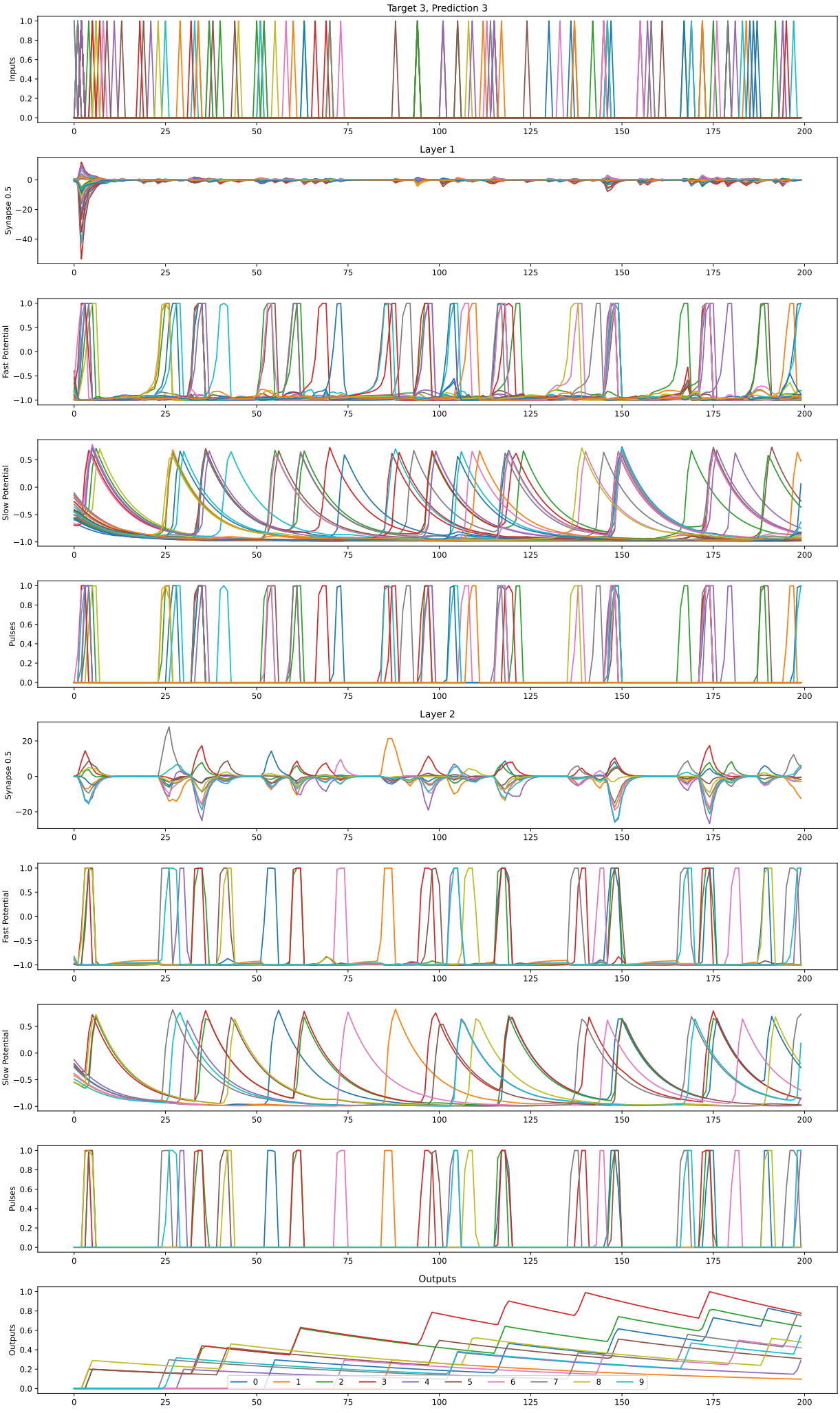
B.4 type 2 latency coding

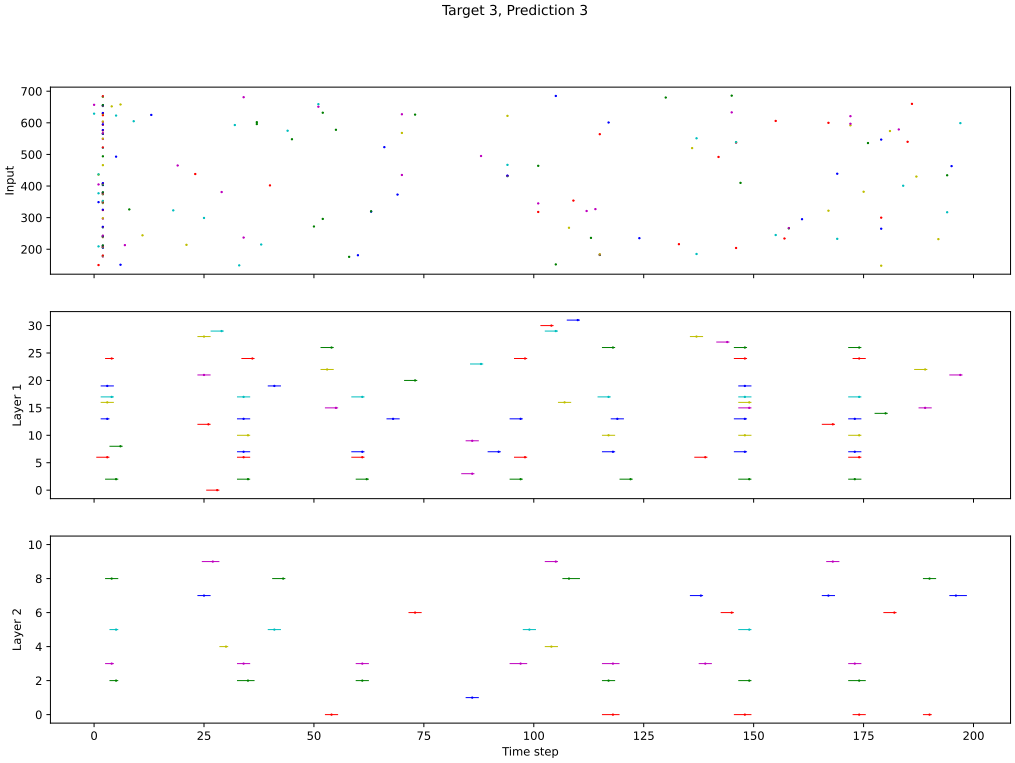

Target 2, Prediction 2
Inputs
Layer 1
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Layer 2
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Outputs
Outputs
0 1 2 3 4 5 6 7 8 9

Target 2, Prediction 2

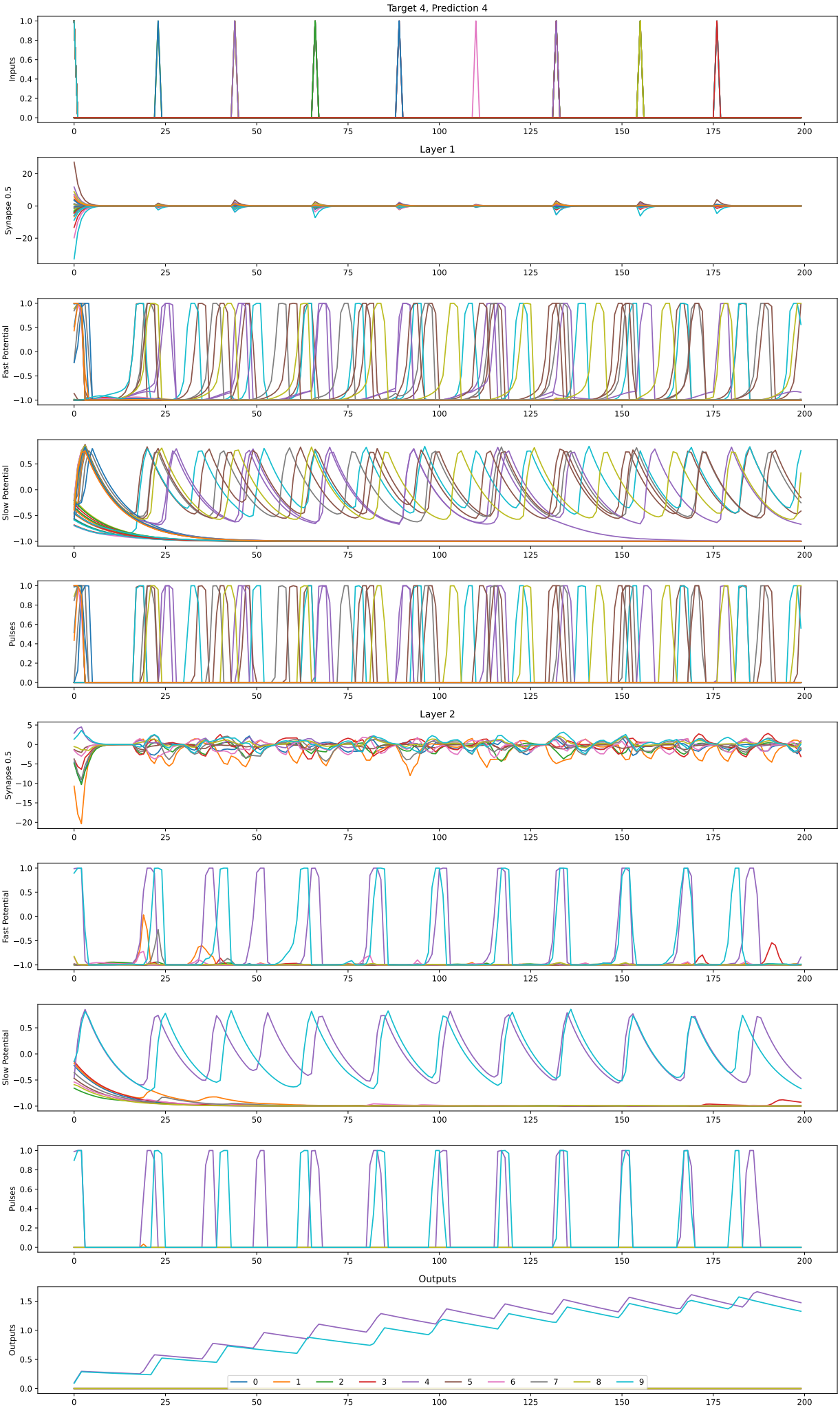


B.5 type 1 latency coding

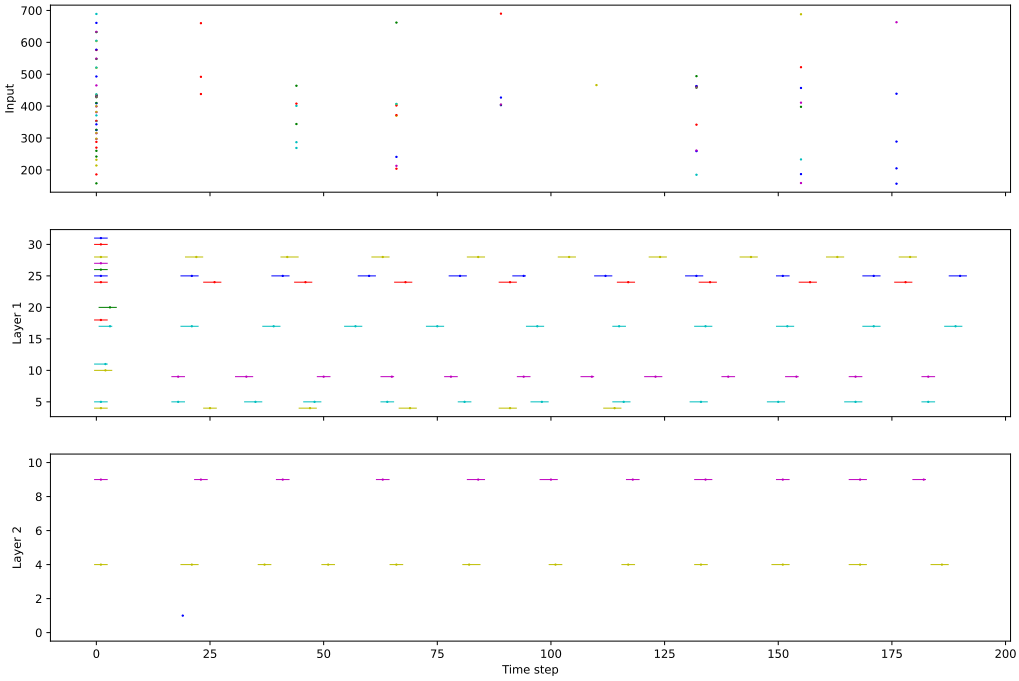

Target 3, Prediction 3
Inputs
Layer 1
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Layer 2
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Outputs
Outputs
0 1 2 3 4 5 6 7 8 9


Target 3, Prediction 3
Input
Layer 1
Layer 2
Time step

B.6 type 2* latency coding


Target 4, Prediction 4
Inputs
Layer 1
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Layer 2
Synapse 0.5
Fast Potential
Slow Potential
Pulses
Outputs
Outputs
0 1 2 3 4 5 6 7 8 9

Target 4, Prediction 4



Appendix C

Spike Statistics For The Trained Network With High Aliasing

In each section of this chapter, the plotted statistics regarding spike width, height, and inter-spike interval for the high-aliasing network are provided. These plots demonstrate trends that are similar to the ones observed in the low-aliasing networks presented in Chapter 6. However, high aliasing does result in narrower spikes and higher spiking frequencies.

C.1 $z_s^{hyp} = 0.5$ rate coded spike statistics

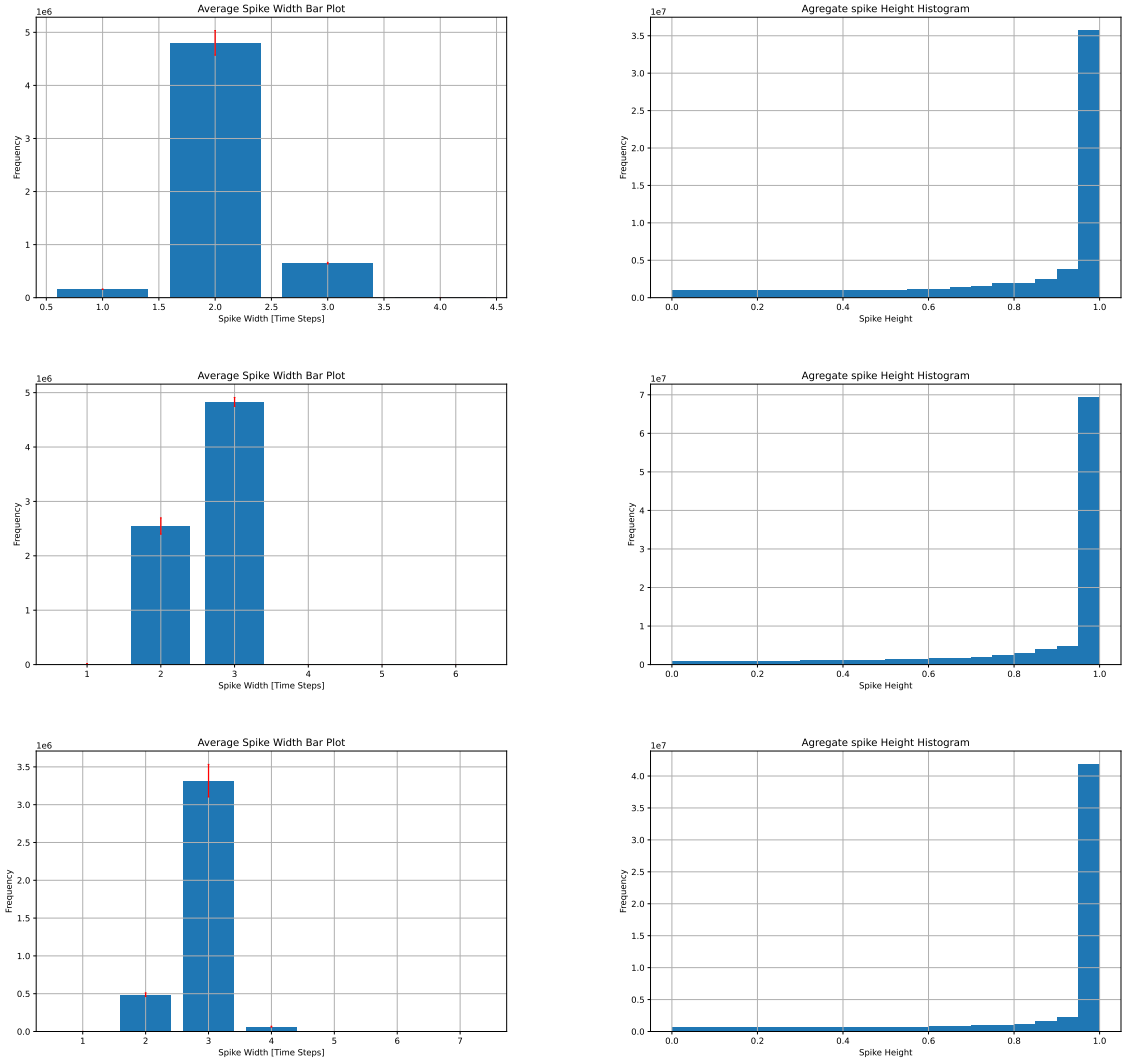
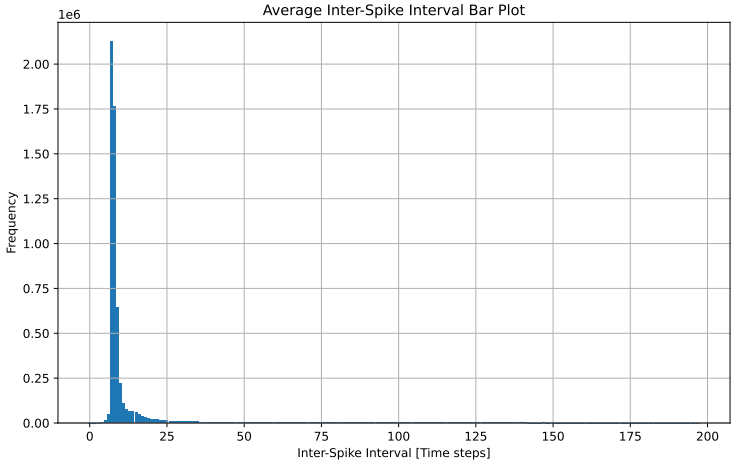
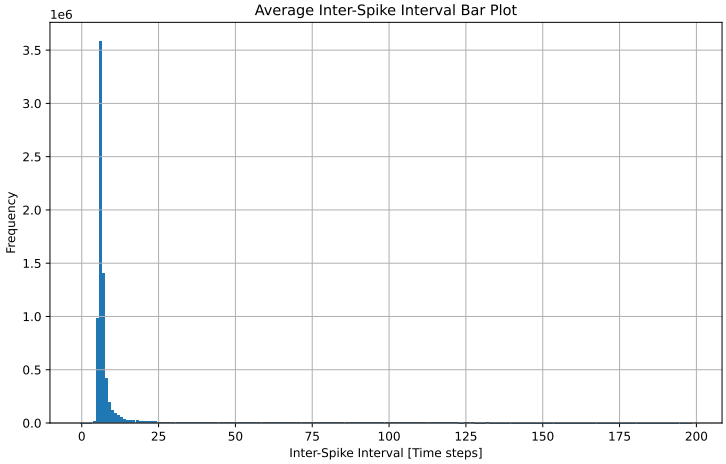


Figure C.1: Spike width and height histograms obtained by inference over the whole rate coded MNIST dataset. From the top row, the used mSRC networks are modulated in type 2, 1, and 2* respectively and with $z_s^{hyp} = 0.5$. Spike width is counted as consecutive positive values whereas height is counted as values above 0 in the voltage trace. For spike width, data is averaged over 5 trained models for spike width where the standard deviation is shown as a red bar on top of each bar. Height data frequency is accumulated over the same 5 models.





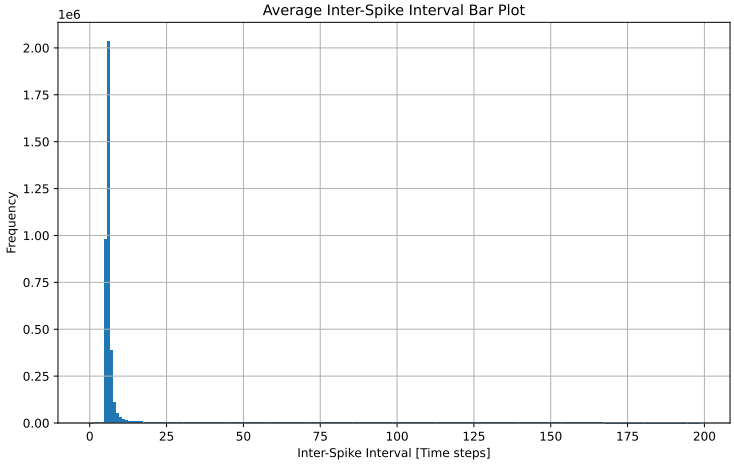


Figure C.2: Inter-Spike Intervals (ISIs) obtained by inference over the whole rate coded MNIST dataset. Starting from the top, the plots correspond to mSRC networks with $z_s^{hyp} = 0.5$ and modulated in type 2, 1, and 2*. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models

C.2 $z_s^{hyp} = 0.5$ latency coded spike statistics

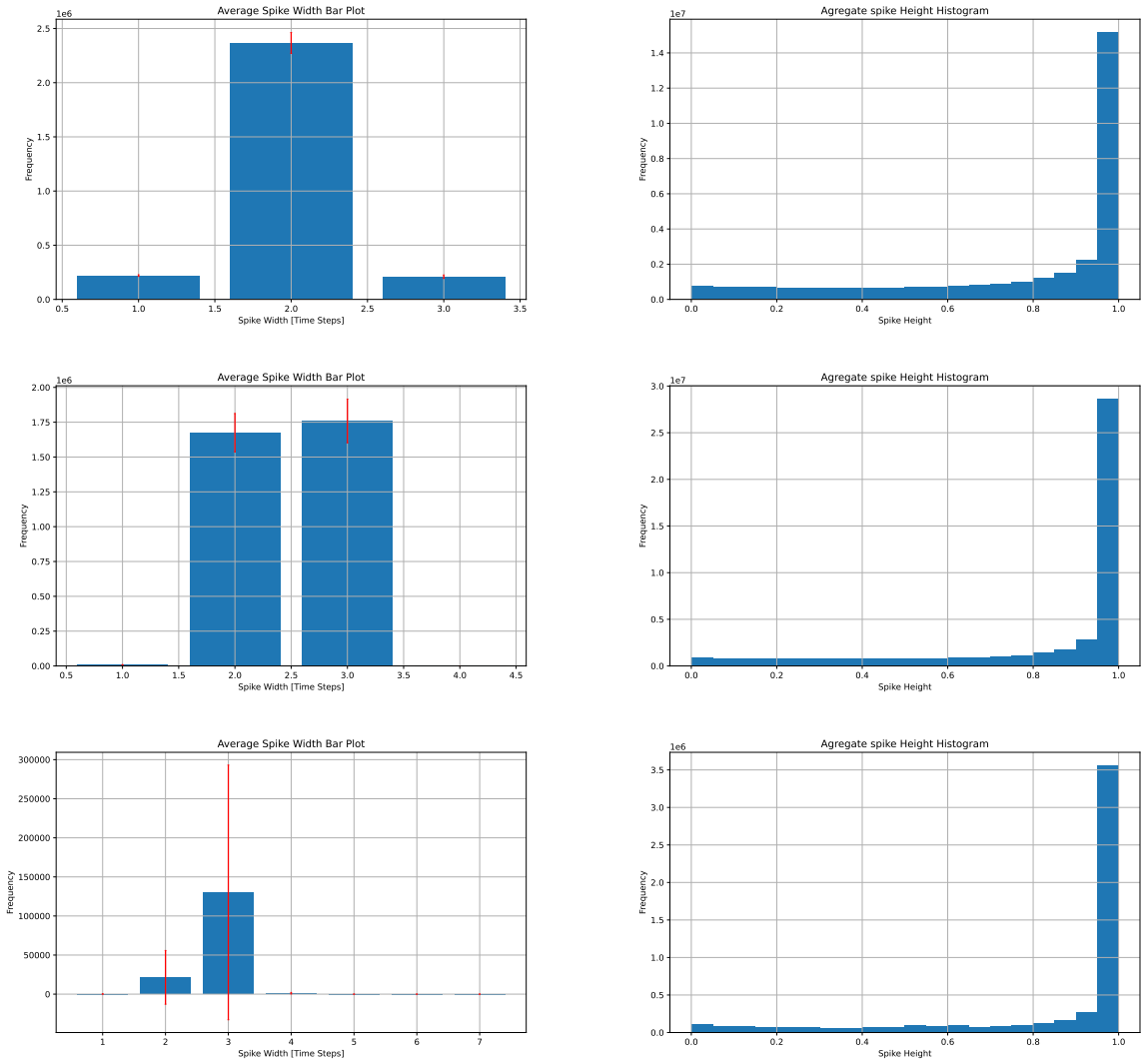


Figure C.3: Spike width and height histograms obtained by inference over the whole latency coded MNIST dataset. From the top row, the used mSRC networks are modulated in type 2, 1, and 2* respectively and with $z_s^{hyp} = 0.5$. Spike width is counted as consecutive positive values whereas height is counted as values above 0 in the voltage trace. For spike width, data is averaged over 5 trained models for spike width where the standard deviation is shown as a red bar on top of each bar. Height data frequency is accumulated over the same 5 models.

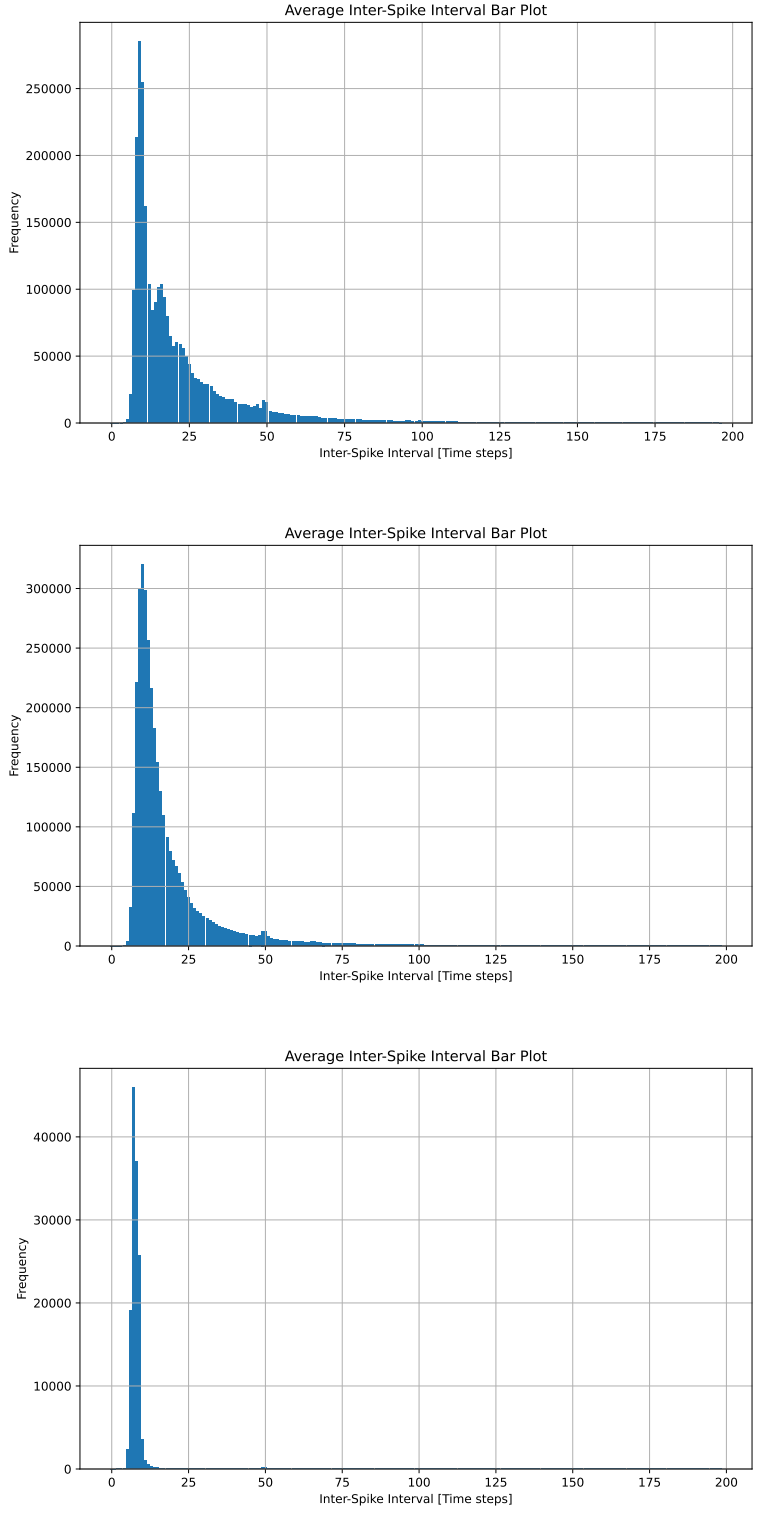


Figure C.4: Inter-Spike Intervals (ISIs) obtained by inference over the whole latency coded MNIST dataset. Starting from the top, the plots correspond to mSRC networks with $z_s^{hyp} = 0.5$ and modulated in type 2, 1, and 2*. ISIs are counted as the number of below-threshold time steps between spikes in the voltage traces. Data is averaged over 5 trained models