LUND UNIVERSITY
Faculty of Science

Developing a spiking neural model of Long Short-Term Memory architectures

Isabella Pozzi

Thesis submitted for the degree of Master of Science
Project duration: 4 months

Supervised by:
Prof. M. Ohlsson, Dr. S. M. Bohte, Dr. D. Zambrano

Department of Physics
January 2018

Evolution has somehow settled on a brain structure that is ideally suited to teasing apart the complexity of the universe.

Brian Wang

Abstract

Current advances in Deep Learning have shown significant improvements in common Machine Learning applications such as image, speech and text recognition. Specifically, in order to process time series, deep Neural Networks (NNs) with Long Short-Term Memory (LSTM) units are widely used in sequence recognition problems to store recent information and to use it for future predictions. The efficiency in data analysis, especially when big-sized data sets are involved, can be greatly improved thanks to the advancement of the ongoing research on Neural Networks (NNs) and Machine Learning for many applications in Physics. However, whenever acquisition and processing of data at different time resolutions is required, a synchronization problem for which the same piece of information is processed multiple times arises, and the advantageous efficiency of NNs, which lack the natural notion of time, ceases to exist. Spiking Neural Networks (SNNs) are the next generation of NNs that allow efficient information coding and processing by means of spikes, i.e. binary pulses propagating between neurons. In this way, information can be encoded in time, and the communication of information is activated only when the input to the neurons change, thus giving higher efficiency.

In the present work, analog neurons are used for training and then they are substituted with spiking neurons in order to perform tasks. The aim for this project is to find a transfer function which allows a simple and accurate switching between analog and spiking neurons, and then to prove that the obtained network performs well in different tasks. At first, an analytical transfer function for more biologically plausible values for some neuronal parameters is derived and tested. Subsequently, the stochastic nature of the biological neurons is implemented in the neuronal model used. A new transfer function is then approximated by studying the stochastic behavior of artificial neurons, allowing to implement a simplified description for the gates and the input cell in the LSTM units. The stochastic LSTM networks are then tested on Sequence Prediction and T-Maze, i.e. typical memory-involving Machine Learning tasks, showing that almost all the resulting spiking networks correctly compute the original tasks.

The main conclusion drawn from this project is that by means of a neuronal model comprising of a stochastic description of the neuron it is possible to obtain an accurate mapping from analog to spiking memory networks, which gives good results on Machine Learning tasks.

Acronyms and Abbreviations

AAN	*Adaptive Analog Neuron*
ANN	*Analog Neural Network*
ASN	*Adaptive Spiking Neuron*
CEC	*Constant Error Carousel*
FFNN	*Feed Forward Neural Network*
ISI	*Inter-Spike Interval*
LHC	*Large Hadron Collider*
LIDAR	*LIght Detection And Ranging*
LIF	*Leaky Integrate-and-Fire*
NN	*Neural Network*
PSC	*Post Synaptic Current*
QNN	*Quantum Neural Network*
ReLU	*Rectified Linear Unit*
RNN	*Recurrent Neural Network*
RTRL	*Real-Time Recurrent Learning*
SRM	*Spike Response Model*
LSTM	*Long Short-Term Memory*
SNN	*Spiking Neural Network*

Contents

Chapter 1

Introduction

The purpose of the present project is to study a biophysical model of spiking neurons that compromises between the amount of biological details included and efficiency in the implementation, in order to develop a spiking neural model able to solve tasks like speech recognition and working cognitive memory, and to be potentially used in Physics applications. A motivation for the importance of having spike-generation mechanism at the heart of a computational model relies on the fact that it was proved, by evaluating the information flow in terms of entropy [1, 2], that spiking neurons are characterized by high coding efficiency, higher than for standard artificial Neural Networks (NNs). Here, the stochastic nature of the spike-firing mechanism typical of biological neurons is applied and tested, in order to see if it is functional.

In this chapter, after an introduction to the world of artificial NNs and in particular the architectures and neurons involved in this project (i.e. Long Short-Term Memory units and spiking neurons), the most relevant spiking neural models are presented, and subsequently the state of the art concerning the relatively new discipline of Neuromorphics and the relationship between spiking networks and the development of Physics research is provided.

1.1 Artificial Neural Networks

Artificial NNs can be defined as computing systems based on a collection of connected units and inspired by the structure and the functioning of the brain. These systems have many capabilities, such as function approximation, time series prediction, object classification, and thanks to their versatility have been used in many circumstances. Historically, the implementation of the so-called *perceptron*, one of the first links between biological neurons and mathematics and one of the first models of artificial neurons, gave rise to the first artificial NNs[3].

In general, a NN comprises an input layer, one or more (necessary for the branch of Deep Learning) hidden layers, and an output layer. Each layer is composed of one or more *nodes*, i.e. neurons, and the neurons are linked through *synapses*, i.e. weighted connections. When an input, which can be of different forms, is fed to the network, this will return an output, which will depend on some factors, such as the values of the weights of the connections.

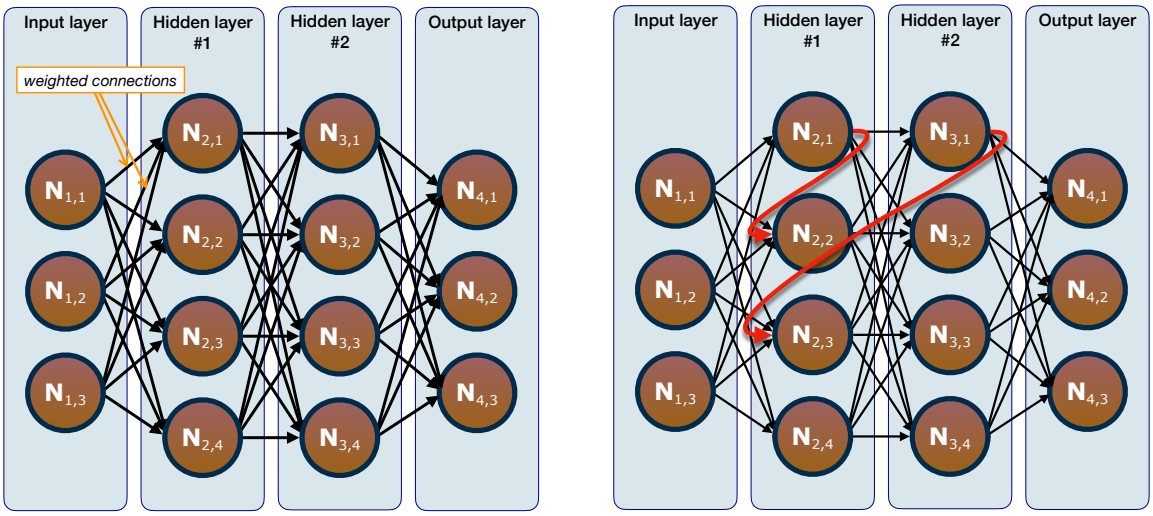

Figure 1.1: *Examples of artificial NNs: (left panel) FFNNs and (right panel) RNNs.*

The first generation of NNs comprises of layers in which the output of a layer is fed as input to the subsequent one, and thus the information flows uniquely in one direction. This type of networks are called Feed Forward Neural Networks (FFNNs), an example of which is given in the left panel of Figure 1.1. Nowadays, perceptrons are considered being a subcategory of FFNNs, since they are algorithms for learning binary classifiers and characterized by having an input vector and a weight vector.

In order to adapt a network to a particular task, in fact, it has first to learn it. This is done by means of training procedures, which can be performed with different approaches, such as *supervised learning* (i.e. expected values and obtained values are compared, and this residual error is iteratively minimized) and *reinforcement learning* (i.e. a learning agent sequentially selects actions, observes rewards, and aims to maximize the total reward over a period of time). Since these two categories of learning processes interest the present project, more details about them are given in Chapter 3.

What makes the learning possible is that the output of the network changes if the weights of the connections are changed. Thus, the learning process consists of iteratively calculating (and reducing) an output error as a function of the weights of the connections. In more detail, each iteration corresponds an update of the synaptic weights following the so-called *gradient descent algorithm*: steps along a *cost function* are taken in the direction towards which its gradient decreases. The cost (or loss, or objective) function for calculating the output error can be chosen among many possible functions, such as the Mean Squared Error (MSE) function and the cross-entropy error function, which was used for this project.

Another key-concept in artificial NNs is a function assigned to each node and that determines the extent with which the change of the network weights gives a change in the output: it is the so-called *transfer function* of the neurons. This function maps the input received by the neuron with the output it will generate. Originally, perceptrons were characterized by having a step function as transfer function, but what one ideally wants for a neural network is that small changes in the weights cause only small changes in the output, and such a non-continuous function was not ideal. For this reason, a *sigmoid* function was introduced as neuronal transfer function, which can be seen as a smoothed-out version of a step function, and for

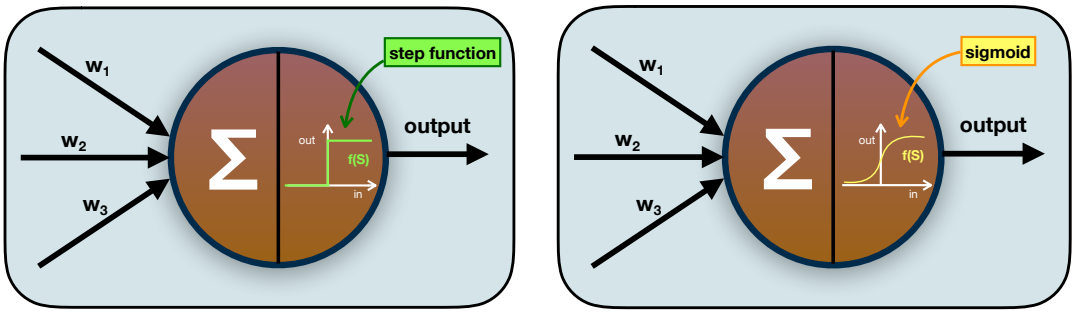

Figure 1.2: *Perceptron (left) and sigmoid neuron (right). The weighted inputs are collected and an output is generated by means of a transfer function.*

its smoothness (i.e. continuity) the sigmoid is commonly used in artificial NNs. In Figure 1.2 a schematic depiction of a perceptron and of a sigmoid neuron is given.

During the learning process, in order to compute the gradient of the cost function, the output error is first calculated and then distributed back through the network layers by means of the so-called *backpropagation algorithm*. In this way, the weight of each connection can be assessed and updated. When the cost function has reached a minimum, the training algorithm has succeeded. However, this procedure is not trivial, and it can present obstacles in many situations. For instance, for the hidden layers of a network, when their number increases, even if intuitively it would seem to enhance the potentiality of the network, it induces complications in the backpropagation algorithm. In particular, it can mean either for the early layers to learn more slowly than the late ones, or vice versa. These phenomena are due to a certain instability of the gradient in deep networks and are known as *vanishing-* and *exploding-gradient problem*, respectively. In order to control this aspect, it is possible to choose a cost function and a *transfer function*, i.e. the function that describes the output of a single node given the input received, which make the gradient less unstable[4, 5].

In general, the backpropagation algorithm can be used not only for FFNNs, but also when more complicated architectures are considered, such as networks with feedback loops. These different patterns for the information flow in which neurons can send their output to neurons of previous, same, or subsequent layers are employed in the so-called Recurrent Neural Networks (RNNs)[6]. An example of RNNs can be seen in the right panel of Figure 1.1. These are considered a particularly important class of NNs, since they include the time dimension in the network, not contemplated in the standard architecture. In fact, by discretizing time in steps of fixed length, the RNNs can be trained to remember information and use it at a later point in time. Schematically, RNNs are represented as FFNNs with repeated weights and units (i.e. the recurrent ones), and that is why FFNNs can be seen as a special case of RNNs. However, these networks are very hard to train because an error has to be backpropagated in time. These difficulties derive from the fact that the derivative of the activation function of the same neuron needs to be applied for multiple time steps and, as mentioned earlier, the gradient in deep neural nets is not stable. This leads to an increased gradient instability for RNNs and, for this reason, while RNNs should theoretically be able to connect information even when the gap between the relevant information and the point where it is needed becomes very large, in practice that was found to be not true[7, 8].

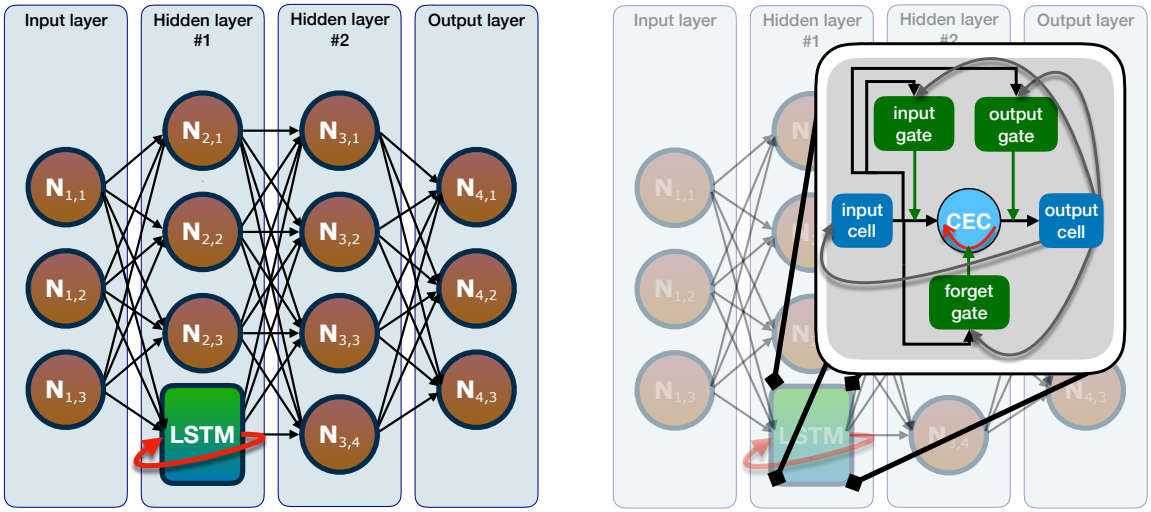

Figure 1.3: *Examples of LSTM networks (left panel) and of LSTM units (right panel).*

Among the many types of RNNs, one of the most important and popular (and the one included in this project), uses a particular memory unit called Long Short-Term Memory (LSTM)[9], which, due to gating structures, avoids the just-mentioned learning problem for short-term memory, i.e. for which the representations of recent input events are stored in form of activations. An example of networks including such architecture is shown in the left panel of Figure 1.3.

1.1.1 Long Short-Term Memory architectures

As anticipated earlier in this chapter, RNNs present a problem in using backpropagation. In fact, the backpropagated error over time depends exponentially on the size of the weights, leading to exploding and vanishing gradient issues. For this reason, it was necessary to come up with an architecture designed to overcome these obstacles hampering the training procedure.

LSTM is an architecture that allows the incorporation of memory in a network and, in particular, to handle the learning of long-term dependencies[9]. It consists of a group of neurons having different behaviors, namely a memory cell which (1) preserves its state over time, hence making it possible for information to persist, and (2) allows constant error flows (i.e. the activation function is an identity function, hence the derivative is always equal to one, hence it does not affect the backpropagation), and gates which control the information flow into and out of the memory cell. Thanks to this structure and in particular to the design of the memory cell, which is the central feature of LSTM and called Constant Error Carousel (CEC), the aforementioned errors can be avoided. Different behaviors are obtained by tuning some parameters. In particular, to make a neuron behave like a gate, it is important to assign it an appropriate transfer function, which has been a crucial point of attention in the course of the present project.

Many variants of the original LSTM unit exist. The most popular are composed of three gating units (input, forget, and output, which control the information flowing into, persisting in, and exiting from the LSTM, respectively), a CEC, and an output activation function[10], as shown in the right panel of Figure 1.3. The forget gate, in particular, by discarding non-relevant and previously stored information (i.e. the

new state of the CEC is obtained by multiplying the previous state by a factor $\in [0, 1]$ which depends on the information flowing from the input cell to the forget gate), is a key feature of LSTM for continually running networks.

1.1.2 Spiking Neural Networks

The idea of artificial NNs finds its roots in one of the most efficient learning and control system known: the brain. In fact, since its neuronal activity mostly depends on the extent with which the input changes, the information flow in the brain is characterized by a sparse nature[11], which translates into energy efficiency.

However, with time the link between biological and artificial NNs has become increasingly tenuous. Nowadays, the related scientific research is mainly oriented towards finding NNs which give higher efficiency and are easy to implement[12]. Nevertheless, scientists have recently started to consider the idea of using networks based on mathematical models of the brain with the goal to achieve its level of efficiency, i.e. the SNNs[13, 14].

In 1952 the first scientific model of a spiking neuron was proposed by Hodgkin and Huxley[15], based on experiments performed on the giant axon of the squid. They succeeded in measuring the currents leading to *action potentials*, i.e. the propagating change in membrane potential controlling the neuronal dynamics, and their mathematical model constitutes the basis for a very detailed neuronal description. Other models (some of which are presented in the next section) were subsequently developed by reducing the amount of details in order to increase the computational efficiency and implementation of the networks, and thus to favor their use in Machine Learning tasks.

Relying on this idea of action potential propagation for passing information between nodes, SNNs are networks in which the neurons, instead of constantly communicating the same piece of information, send each other *spikes* of signals, i.e. small electrical pulses (the biological ones have amplitude of 100 mV and duration 1-2 ms) corresponding to the propagation of action potentials, at either regular or irregular intervals, separated by at least a minimal distance referred to as the *absolute refractory period* of the neuron[1]. Once the spike train is received, it can cause a change in the *membrane potential*, i.e. a quantity that simulates the potential difference between the interior of the biological neuron and its surroundings, of the next neuron. At this point, if the potential of the receiving neuron reaches a critical value above the *resting potential*, i.e. the value of the membrane potential when no inputs were received for some time (approximately -65 mV), this neuron, in turn, starts generating spikes. The critical voltage for spike initiation is commonly described by a *threshold* ϑ, and the moment of threshold crossing defines the *firing-* or *spike-time*. In Figure 1.4 a schematic representation of the functioning of two spiking neurons is given.

The neuron sending spikes can thus be seen as an encoder of information, which then has to be decoded by the receiving neuron. In these regards, it is important where, in the context of this spike-mechanism picture, the biggest amount of information resides. The shape of the pulse does not change while it propagates, and different isolated spikes have the same shape, hence the form cannot be used to retrieve the

[1] *http://www.physiologyweb.com/lecture_notes/neuronal_action_potential/neuronal_action_potential_refractory_periods.html*

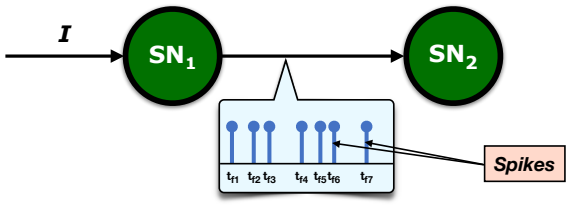

Figure 1.4: *Depiction of spiking neurons and their characteristic event-based coding.*

encoded information. Instead, other information holders have been studied, such as the firing rate, i.e. the number of spikes per time interval reaching the subsequent neuron (*rate coding*[16]), or the spike time, which gives to the information flow in SNNs an event-based nature (*temporal* or *spike-time coding*[17]).

The spiking nature of these neurons gives rise to a fundamental problem in the implementation, since the spikes are not differentiable, thus hampering the training and backpropagation processes (that can usually be performed due to the differentiability of each component of the networks). Training algorithms for spiking networks have been studied, but they are much less efficient than the training procedure for the standard artificial networks. A solution to this issue–and the one implemented in the present work–is to use analog neurons for training and subsequently replacing them with spiking neurons, with the condition that the activation function of the analog neurons has to approximate the behavior of the spiking neuron.

1.2 Spiking Neural Networks and Physics

There are many links between Physics and artificial NNs, for example in the last few decades models for NNs based on Quantum Mechanics have been studied. These nets, known as Quantum Neural Networks (QNNs), are built using the principles of quantum information processing, which translates into taking into account not only the amplitude but also the phase of the signal[18]. The ongoing research focuses on exploiting the computational and statistical complexity provided by quantum perceptron models[19]. The ability of QNNs in image compression was tested and it was found that their efficiency was comparable if not greater than standard neural nets[20].

In the same way QNNs can be seen as the point of contact between Quantum Mechanics and Deep Learning, SNNs are the meeting point between Biophysics and Deep Learning. More in general, Physics and SNNs are interlinked with each other, since the former is deeply embedded in the theory underlying the spiking neural models and the implementation of SNNs for data gathering, storage, and analysis could lead to higher efficiency time- and computational-wise, useful in many branches of Physics research.

1.2.1 Biophysics of the Spiking Neural Model

In Physics, one has to distinguish between different scales, for which distinct mathematical formulations hold, e.g. the very Newton's second law, $F = ma$, is very efficient in describing systems at big scales but, when going down to small (i.e. atomic) scales, it does not hold anymore, and that is when a different formalism, i.e. quantum

mechanics, comes to the rescue. In the same way, different biophysical models for the neuron exist, which can be classified depending on both the amount of details included and the computational efficiency of their implementation. In particular, the very foundation of all the spiking neural models developed so far comes from concepts borrowed from Physics, and thus they can be seen as different physical models. In this section some models are presented, going from the best in terms of completeness and amount of details included, through more simplified phenomenological ones, to the most efficient when it comes to implementation.

Hodgkin-Huxley

The most detailed description of neuronal dynamics is given by the Hodgkin-Huxley model[15], as mentioned in the previous section. It can characterize in detail different types of synapses and the spatial geometry of each neuron by means of a set of differential equations describing the dynamics of the currents flowing through the semipermeable cell membrane via ionic channels, leading to action potentials. The entire neuronal dynamics, in fact, stems from electrical potentials generated by a difference in ion concentrations between the inside of the neuron and the surrounding environment.

In principle, a complete biophysical model for the neuron can be built if both the channels present and their respective parameters are known. Originally, Hodgkin and Huxley included in their theory only two ionic channels, i.e. sodium and potassium, and a leakage channel. Under this assumption, the Hodgkin-Huxley model is represented by a system of four nonlinear differential equations:

$$\begin{cases} I(t) = C\dfrac{\mathrm{d}u}{\mathrm{d}t} + g_{\mathrm{Na}}m^3h(u - E_{\mathrm{Na}}) + g_{\mathrm{K}}n^4(u - E_{\mathrm{K}}) + g_{\mathrm{L}}(u - E_L\mathrm{L})\,, \\ \dot{m} = -\dfrac{1}{\tau_m(u)}[m - m_0(u)]\,, \\ \dot{n} = -\dfrac{1}{\tau_n(u)}[n - n_0(u)]\,, \\ \dot{h} = -\dfrac{1}{\tau_h(u)}[h - h_0(u)]\,. \end{cases} \tag{1.1}$$

The first equation describes the current at the level of the membrane. It is composed of a contribution representing the charging current and three terms representing the current flowing through the ion channels (the subscript L refers to unspecific leakage channels), regulated by some gates. The parameter g_i and the quantity E_i are, respectively, the conductance and the voltage relative to the channel i. The other three equations are the evolution of the gating variables m, n, and h, with m and h controlling the evolution of the sodium channels, and the gating variable n controls the potassium channels. A more detailed procedure leading to System 1.1 is given in the Appendix.

This model can describe many processes part of the neuronal dynamics. For example, if immediately after a spike another input is delivered to the Hodgkin-Huxley model, evidence of *refractoriness* can be observed. This is due to a lowering of the membrane potential after the spike is emitted, hence the charge needed to fill

the gap between the value of the membrane potential and the critical point is bigger than the initial value. The intrinsic refractoriness of the neuronal signal represents a crucial feature in spike-based communication that needs to be taken into account when using adapting spiking networks, and it will thus be mentioned again in the course of this thesis.

Even though their description is outside the scope of the present work, it must be mentioned that many other neuronal dynamics aspects can be correctly described by the Hodgkin-Huxley model. However, despite its accuracy and robustness, the difficulty in visualizing and analyzing the behavior of high-dimensional systems of non-differential equations it is not insubstantial. For this reason, a reduction to two dimensions can be conveniently studied, since for systems of two differential equations a mathematical analysis is possible.

FitzHugh-Nagumo

The first proposing to study the generation of action potentials by reducing the dimensions of the neuronal model from four to two were FitzHugh and Nagumo[21, 22]. By applying some approximations (for more details, see the Appendix), the neuronal dynamics can be described by the following two differential equations:

$$\begin{cases} \dfrac{\mathrm{d}u}{\mathrm{d}t} = \dfrac{1}{\tau}[F(u,w) + RI]\,, \\ \dfrac{\mathrm{d}w}{\mathrm{d}t} = \dfrac{1}{\tau_w} G(u,w)\,, \end{cases} \tag{1.2}$$

with R being the resistance and some functions F and G, such that G interpolates between $\mathrm{d}n/\mathrm{d}t$ and $\mathrm{d}h/\mathrm{d}t$.

Following this path, different mathematical formulations for the functions F and G were proposed. FitzHugh and Nagumo defined them as

$$\begin{aligned} F(u,w) &= u - \frac{1}{3}u^3 - w\,, \\ G(u,w) &= b_0 + b_1 u - w\,, \end{aligned} \tag{1.3}$$

obtaining sharp pulse-like oscillations reminiscent of action potentials.

In order to visualize the temporal evolution of the variables (u, w) the two-dimensional phase plane analysis can be used. The neuron, in fact, being a system that evolves over time, can be considered as a dynamical system. By integrating the Equations 1.2 over time, the evolution of the state of the system (i.e. the neuron) can be studied. By performing such a study, the presence of three fixed points in the phase portrait of the FitzHugh-Nagumo model can be highlighted, due to intersections of the u- and w-nullclines (i.e. the points such as $\dot{u} = 0$ and $\dot{w} = 0$, respectively). By solving an eigenvalue problem, the study of the stability of these points can be performed. Hence, by means of the Poincaré-Bendixson theorem, the existence of a limit cycle can be proven, leading to the interpretation of the neuron as an oscillatory system: following the trajectories obtainable by injecting a positive I into the system, the voltage follows rapid changes, which can be seen as a train of spikes. In Figure 1.5 the nullclines and the trajectory, i.e. a limit cycle (resulting from the integration over time of a dimensionless version of Equations 1.2, see [23] for

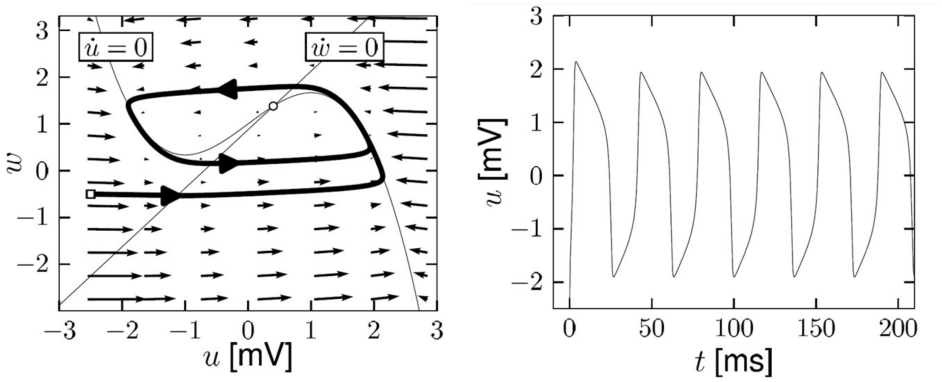

Figure 1.5: *Phase plane analysis for the FitzHugh-Nagumo model with dimensionless variables. Left panel: nullclines for I=2 and trajectory (thick line); right panel: voltage time course along the trajectory shown in the left panel.[23]*

further details), of the FitzHugh-Nagumo model for a positive input are shown (left panel), together with the voltage time course of the trajectory (right panel), showing a spiking trend.

Leaky Integrate-and-Fire (LIF)

The integrate-and-fire family of models can be interpreted as the opposite of the Hodgkin-Huxley model, since they are stripped of all the biology inspired details and very easily implementable, thereby representing the most basic neuronal models. The simplest version of the Leaky Integrate-and-Fire (LIF)[24, 25, 26, 27] model is constituted of a single linear differential equation describing the evolution of the membrane potential and a threshold for spike firing.

The name derives from the fact that, as a first approximation, the dynamics of neurons can be described as a summation (or integration) process. In fact, for low firing rates the total change of potential in the postsynaptic neuron is given by the summation of the changes of potential caused by each individual presynaptic neuron. In a slightly more general version of this model (i.e. the nonlinear LIF), when the potential u crosses the threshold ϑ_{reset}, a spike is generated, defining the firing time t_f, and, instead of having a relaxation variable like in the previous models, here the membrane is reset after every spike.

However, neglecting the more detailed cases (not covered here) in which the phenomenon of adaptation is included in the picture, the linear LIF model can be represented by an ensemble of discrete physical components: a capacitor and a resistor placed in parallel and driven by a current. In particular, the membrane is seen as a capacitor with a finite leak resistance. By means of the law of current conservation, the input current $I(t)$ is split in two components: a resistive current $I_R = [u(t) - u_{\text{rest}}]/R$ and a current $I_C = \mathrm{d}q/\mathrm{d}t = C\mathrm{d}u/\mathrm{d}t$ that charges the capacitor, giving the following:

$$\begin{aligned} I(t) &= I_R + I_C \\ &= \frac{u(t) - u_{\text{rest}}}{R} + C\frac{\mathrm{d}u}{\mathrm{d}t}\,, \end{aligned} \tag{1.4}$$

and by introducing the membrane time constant $\tau_m = RC$ the standard formulation[23]

is obtained:

$$\tau_m \frac{\mathrm{d}u}{\mathrm{d}t} = -[u(t) - u_{\mathrm{rest}}] + RI(t)\,, \tag{1.5}$$

which is a linear differential equation describing a RC-circuit with the resistor and the capacitor arranged in parallel, or *equation of a passive membrane*. The solution assuming that for $t > 0$ the input vanishes and with initial condition $u(t_0) = u_{\mathrm{rest}} + \Delta u$ is given by[23]:

$$u(t) - u_{\mathrm{rest}} = \Delta u \exp\left(-\frac{t - t_0}{\tau_m}\right) \quad \text{for } t > t_0\,. \tag{1.6}$$

This shows that with no input the membrane relaxes back with a decay regulated by τ_m (usually equal to approximately 10 ms) to the resting value. In this framework, $\vartheta_{\mathrm{reset}}$ can be interpreted as the minimal voltage necessary to cause a spike.

This one-dimensional model is being used for doing simple computation with SNNs but, since it does not take into account of adaptation, it is considered oversimplified in the sense that it cannot describe the firing mechanism of biological neurons at all, in contrast to the rest of the previously-presented models.

Spike Response Model (SRM)

Another mathematical description of neuronal activity, and the one underlying the model used in thesis, is given by the so-called Spike Response Model (SRM)[28, 29]. It can be derived from the LIF by integrating a series expansion[14], however it differs from the other models presented in this section because, instead of making the variables changing as a function of the voltage, here they are described by the spike time and in particular by the time passed since the last output spike. The SRM simplifies the spike generation mechanism while keeping a very rich description in terms of neuronal refractoriness and adaptation. In fact, the firing of a spike at time t is obtained whenever the membrane potential reaches–from below–the threshold $\vartheta(t)$, which is time-dependent, or, in formula:

$$t = t_f \qquad \Leftrightarrow \qquad u(t) = \vartheta(t) \quad \text{and} \quad \frac{\mathrm{d}[u(t) - \theta(t)]}{\mathrm{d}t} > 0\,, \tag{1.7}$$

with t_f as the firing time.

In this framework, the variable u describing the membrane potential fully characterizes the state of the neuron and its evolution over time is given by:

$$u(t) = \int_0^\infty \eta(s) S(t-s)\mathrm{d}s + \int_0^\infty \kappa(s) I^{\mathrm{ext}}(t-s)\mathrm{d}s + u_{\mathrm{rest}}\,, \tag{1.8}$$

with $I^{\mathrm{ext}}(t)$ representing an external time-varying stimulating current, u_{rest} the resting potential, $S(t)$ the spike train, κ and η being two kernels controlling the shape of the membrane and of the voltage-dependent output, respectively. More in detail, the membrane shows a linear response to an injected current, and, on the other hand, it takes time for the membrane to decay back to the value of u_{rest} after receiving an input. The two filters κ and η respectively control these two processes.

It was proved that, if some features are added to a more detailed version of this model, the in-vitro experimental signal made of spikes can be fitted with a precision in the order of milliseconds[30]. In Figure 1.6 a depiction of the SRM including adaptation is shown.

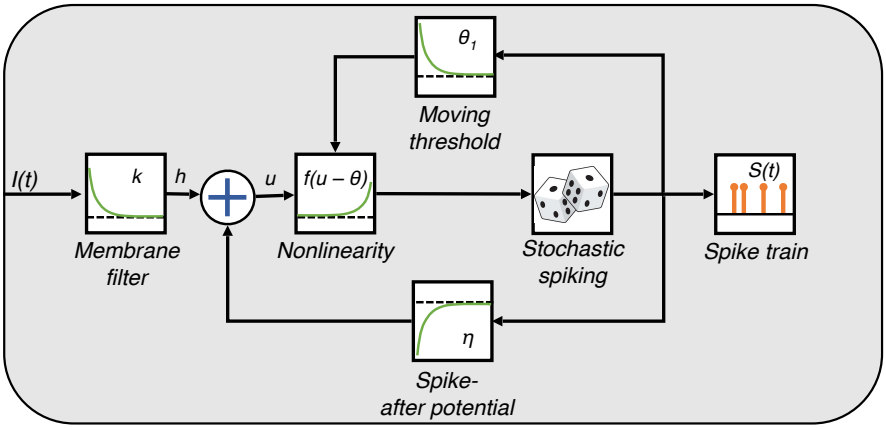

Figure 1.6: *Schematic description of the SRM. The variable h indicates the input potential after being filtered; the other quantities are presented in the text.*

1.2.2 Applications in Physics

From gene mapping to space exploration, the problem of gathering data sets so huge that are difficult to save and analyze concerns many disciplines, as described by Sivarajah et al.[31].

Thanks to a better balance between communication time and computation cost than traditional networks, SNNs have demonstrated great potential in big data analysis[32]. Many applications use RNNs to integrate or store signals through time, but the standard NN computation lacks the natural notion of time (i.e. it is a frame-based method), which makes this approach not appropriate, whilst, on the other hand, SNN computation is time-based. These networks carry advantages particularly tied to parallel computing, a type of computation which has been used to model problems, on top of other scientific domains, in many branches of Physics, such as applied, nuclear, particle, condensed matter, high pressure, fusion, and Photonics[33, 34, 35, 36].

Astrophysical observations, for example, involve the gathering of huge amounts of data[37]. Machine Learning techniques have been already giving major support to this type of applications[38] by effectively building a priori knowledge that can be used to go beyond the deconvolution limit set by the Shannon-Nyquist sampling criterion[39]. A further and more specific example is given by the research conducted at the Large Hadron Collider (LHC)[40] facility, where the Higgs boson was discovered. On July 4, 2012, after collecting and analyzing 500 trillion proton collisions[41], the existence of a boson of mass could be announced. This figure is a prime example of the extent of the necessity for scientists, which typically have to select images showing some desired features and discard the rest, i.e. a practice requiring a lot of manual tuning, to find more efficient solutions for data gathering, storing, and processing. In particular, with regard to the issue of data storage, in 2005 the world's first petabyte-sized database was introduced with the BaBar experiment[42]. Because of some limitations found at that time, however, it was decided to store data in ROOT and that is how today hundreds of petabytes of data worldwide are stored, even though ROOT has been criticized for many aspects, such as the difficulty for beginners, its design, and its implementation[2]. The same issues interest other branches, and in particular it applies to X-ray Crystallography, a technique for determining three-dimensional

[2] *http://insectnation.org/articles/problems-with-root.html*

atomic structures through light-scattering detection. Besides the problem of data storage, for these experiments, in order to recognize the presence of Bragg peaks in images, multiple offline analyses months after the experiment are usually needed to be completed[43]. In these regards, the attractiveness of in-situ fast data storage and analysis offered by SNNs stands out.

Moreover, given the asynchronous nature of spiking LSTM, the implementation of SNNs would bring a major benefit to additional research fields for which the estimation of the correlation between time series is required. Experiments involving the practice of collecting data at different frequencies is, in fact, hampered by the use of a time window by the detectors. Spiking LSTM networks would provide asynchronous data gathering, allowing simple signal decorrelation. This feature would be beneficial also within other fields, such as medical and financial applications [44], but of course scientific research in general would receive a boost from asynchronous detection and computing. An example of the procedures which would take advantage of the implementation of spiking memory networks is given by LIght Detection And Ranging (LIDAR), a remote sensing technique used to probe objects at various scales at very distant positions by detecting the backscattered radiation of pulsed lasers. Distances are thus found by calculating the time span between the moment at which the pulse was sent and the detection time, hence a detection without an internal clock would allow to greatly improve these measurements.

The introduction to SNNs as new computing architectures for handling big data and time series is so promising that a relatively new research branch known as Neuromorphics[45, 46] has been sprouting. Its main goal is to develop chips able to process data the same way our minds do, and, by incorporating these chips into detectors, every imaging facility would benefit of the possibility of computationally efficient in-situ exploration. As Paolo Calafiura (software & computing manager for the LHC ATLAS experiment) said: "The field of neuromorphic computing is very new, so it is hard to say conclusively whether science will benefit from it. But from a particle physics perspective, the idea of a tiny processing unit that is self-contained and infinitely replicable is very exciting".

In these regards, many neuromorphic studies have been emerging. In 2014, the American multinational technology company IBM developed and produced the first neuromorphic chip, called TrueNorth[47], and in 2016 IBM scientists published a paper proving that convolutional networks could run quickly and accurately on said neuromorphic chip[48]. In 2014 the Advanced Processor Technologies Research Group (APT) at the School of Computer Science, University of Manchester, designed a computer architecture named SpiNNaker (*Spiking Neural Network Architecture*), which is planned to be a massively parallel computing platform based on SNNs [49]. Moreover, earlier this year Intel announced a neuromorphic artificial intelligence test chip named Loihi[3], which will be produced in the next few months. The curiosity arising withing the scientific community towards SNNs can be also noticed from the fact that, currently, scientists at Berkley have been testing whether these brain-like architectures have the potential to handle Big Data problems. On a final note, a spiking LSTM networks was implemented by TrueNorth[50], however it does not provide asynchronous data processing, resulting in a less-efficient and less biologically plausible model than the one proposed in this thesis.

[3] *https://newsroom.intel.com/editorials/intels-new-self-learning-chip-promises-accelerate-artificial-intelligence/*

Chapter 2

The model

As stated previously, the scope of this project is to study a model for spiking neurons which could be applied to LSTM networks, in order to exploit the potential of SNNs and make a small step towards their implementation in many scientific branches. The main goal is to study and optimize the conversion from analog to spiking neurons, to obtain spiking working memory networks.

In this chapter, at first, the neuronal model used for modeling the behavior of the spiking neurons involved is described. Subsequently, the problem of finding a transfer function for the analog neurons in order for them to be able to describe how their spiking counterparts communicate information is addressed, with the goal of training the LSTM network using a continuous and derivable function. Specifically, an analytical solution biologically plausible is firstly derived mathematically, in order to study whether this option could bring some advantages, e.g. error reduction, in the storing process of the memory cell of the LSTM units, i.e. the CECs (analyzed in the next chapter). Successively, the procedure leading to another solution (which will be tested in the next chapter) is presented, involving the incorporation of a stochastic behavior in the neuronal model initially described. Finally, since the Machine Learning tasks were used to test only the stochastic network, the adaptive stochastic LSTM unit is presented at the end of this chapter, followed by an overview concerning the theory of the learning algorithms involved in this thesis (i.e. supervised and reinforcement learning).

2.1 Adaptive Spiking Neurons

The spiking neurons that were used for this project are multiplicative Adaptive Spiking Neurons (ASNs) as described in [51] and their analog counterparts, Analog Adaptive Neurons (AANs). This neuronal model is a variant of an adapting SRM that includes fast–multiplicative, instead of additive[52, 53]–adaptation to the dynamic range of input signals, i.e. a central characteristic for obtaining efficient neural coding.

The adaptation is included in two senses: by considering (1) spike-triggered adaptation currents and (2) a dynamical threshold, leading to an increased gap that has to be filled in order for a neuron to emit a spike, thus giving even sparser signals i.e.

more computational gain. This multiplicative version was proven to be able to maintain a high coding efficiency for inputs that vary over several orders of magnitude, unlike its additive counterpart, while still matching neural responses[51].

More in detail, the behavior of the just-introduced ASNs is mathematically described by the following equations[51, 54], expressing the incoming signal (i.e. *incoming postsynaptic current*), the effective received signal, filtered by the neuron (i.e. *input signal*), the *dynamical threshold* determining the critical value the potential difference has to reach for meeting the spike-firing condition, and the *internal state* of the neuron:

$$\text{incoming postsynaptic current:} \quad I(t) = \sum_i \sum_{t_{fi}} w_i \exp\left(\frac{t_{fi} - t}{\tau_\beta}\right), \tag{2.1}$$

$$\text{input signal:} \quad S(t) = (\phi * I)(t), \tag{2.2}$$

$$\text{threshold:} \quad \vartheta(t) = \vartheta_0 + \sum_{t_f} m_f \vartheta(t_f) \exp\left(\frac{t_f - t}{\tau_\gamma}\right), \tag{2.3}$$

$$\text{internal state:} \quad \hat{S}(t) = \sum_{t_f} \vartheta(t_f) \exp\left(\frac{t_f - t}{\tau_\eta}\right), \tag{2.4}$$

where w_i is the weight (synaptic strength) of the neuronal incoming connection, $t_{fi} < t$ denote the spike times of the presynaptic neuron i, and $t_f < t$ denote the spike times of the neuron itself, $\phi(t) = \phi_0 \exp(-t/\tau_\phi)$ is an exponential smoothing filter with time constant τ_ϕ, ϑ_0 is the resting threshold, m_f is the multiplicative variable controlling the speed of spike-rate adaptation, and τ_β, τ_γ, and τ_η are the time constants that determine the rate of decay of $I(t)$, $\vartheta(t)$, and $\hat{S}(t)$ respectively. The neural coding process can then be seen as a sigma-delta modulation[55] (i.e. an encoding process of analog signals into digital signals) for which the receiving neuron decodes the information encoded by the previous neuron by using as starting point the exact times of the spikes.

For this neuronal model, for an ASN receiving a constant input S, the neuron fires a spike whenever:

$$S - \hat{S}(t) > 0.5 \cdot \vartheta(t), \tag{2.5}$$

which can be biologically interpreted as the point at which the different between the outer (S) and the inner ($\hat{S}$) potential of a neuron reaches a critical limit, set as half the value assumed by the membrane potential (ϑ) at that specific moment.

Hence, for our artificial spiking neurons, whenever the spike-firing condition (which was defined in [51]) is met, three things happen: (1) the neuron emits a spike of fixed height h (which, as suggested by [56], is set to the value $h = 1$) to the synapses connecting to the target neurons, (2) a value equal to $\vartheta(t_f)$ is added to the internal state $\hat{S}$, with t_f the spike time, and (3) the threshold is increased by a value $m_f \vartheta(t_f)$.

Some examples of these adaptation processes are depicted in Figure 2.1, with the red (left column) and the purple (right column) curves showing, respectively, how the internal approximation $\hat{S}(t)$ and the threshold $\vartheta(t)$ vary over time when a constant input current is injected into an ASN, for different sets of values of certain parameters of the neuron; in the first row, the plots were obtained by setting biologically plausible values for the time constants, namely $\tau_\eta = 50$ ms and $\tau_\gamma = 15$ ms, and as resting

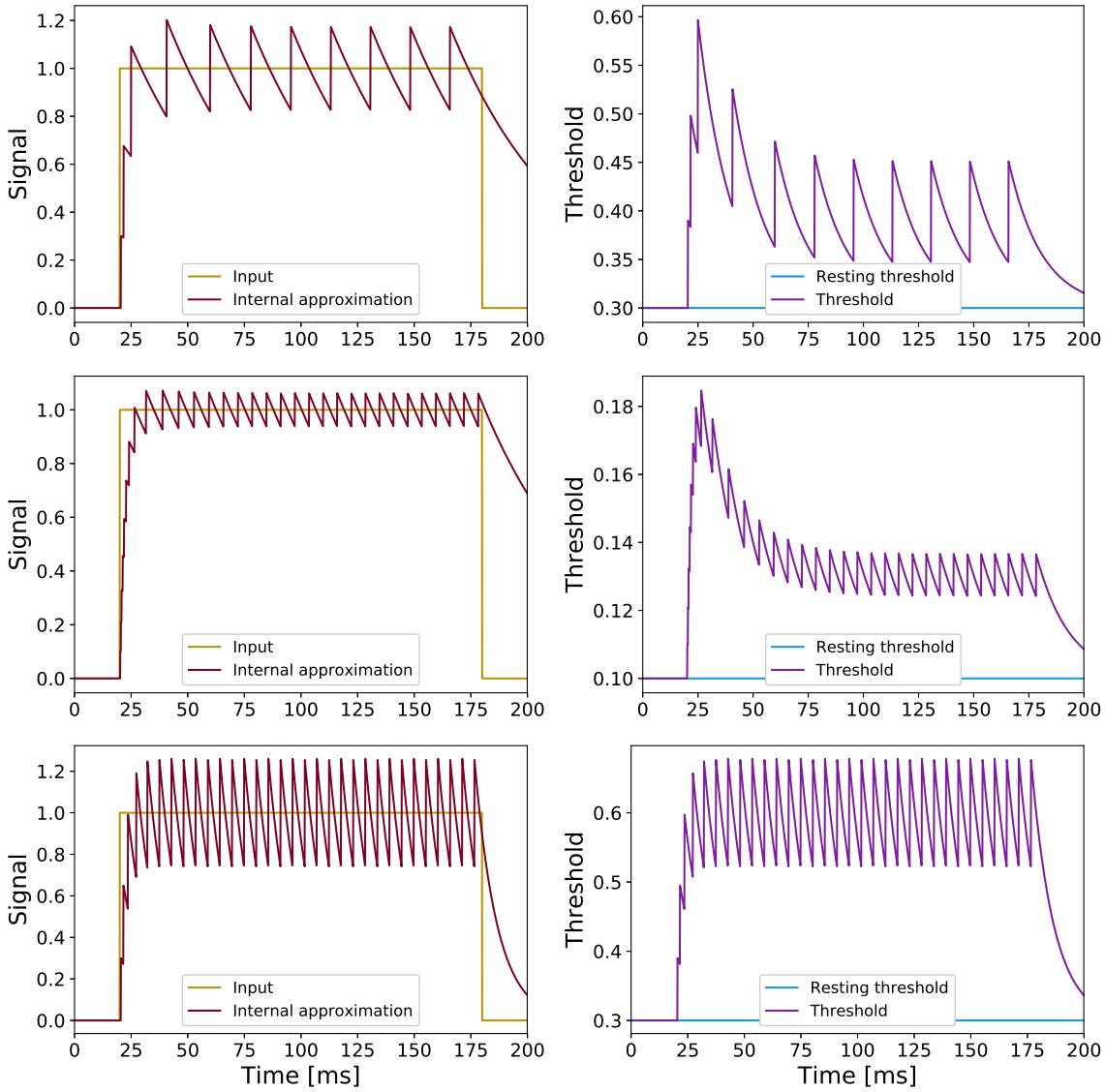

Figure 2.1: *Examples of adaptation dynamics for the ASNs. The internal state together with the input current (left column) and the threshold together with the resting threshold (right column) are plotted as functions of time for different values of some of the neuronal parameters: (top panels)* $\tau_\eta = 50$ *ms,* $\tau_\gamma = 15$ *ms,* $\vartheta_0 = m_f = 0.3$*; (middle panels)* $\tau_\eta = 50$ *ms,* $\tau_\gamma = 15$ *ms,* $\vartheta_0 = m_f = 0.1$*; (bottom panels)* $\tau_\eta = \tau_\gamma = 10$ *ms,* $\vartheta_0 = m_f = 0.3$*.*

threshold and multiplicative factor $\vartheta_0 = m_f = 0.3$; in the second row of plots the threshold and multiplicative factor were changed to $\vartheta_0 = m_f = 0.1$, while the last row represents very fast neuronal adaptation dynamics, obtained by setting $\tau_\eta = \tau_\gamma = 10$ ms while keeping the original values of $\vartheta_0 = m_f = 0.3$. For all the curves, the first few spikes happen very close to each other. This is the *transient phase* of the neuron, i.e. the phase during which the neuron adapts to the signal before getting stable after being silent for some time, and it could represents a source of error when spiking neurons are switched with analog neurons. The signal that needs to be communicated by the neuron is in fact the part of the signal after the transient phase. Precisely, the spikes fired during the transient phase can cause a sudden increase in the signal when perceived by the next neuron, and this is one of the major problems

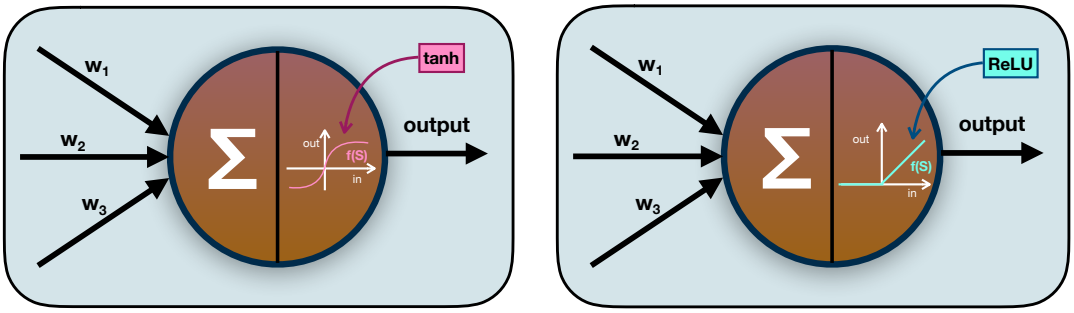

Figure 2.2: *Tanh (left) and ReLU (right) neuron. The weighted inputs are collected and a different output is generated depending on the choice of the transfer function.*

encountered, e.g., in the LSTM conversion used in this project, as it will be shown in Chapter 4.

Overall, still by looking at the plots in Figure 2.1, it is noticeable that by reducing the resting threshold and the multiplicative factor at the same time (i.e. going from the first to the second row of plots), the variance of the signals decreases but the firing rate increases (i.e. the spikes are less sparse), whilst the trend of the adaptation does not change, giving overall a more precise signal encoding. Instead, by decreasing the time constants (i.e. compare the first and the third rows of plots) the speed of the adaptation changes, specifically the adaptation happens quickly leading to an increase in the firing rate. It must be mentioned that, in the present study, τ_β (i.e. the time constant regulating the decay of the incoming postsynaptic current) is always considered to have the same value as τ_η (i.e. the decay constant of the internal state of the neuron), in order to allow for the neuron receiving the signal to be able to decode the signal sent from the previous neuron. The trends depicted in the figure represent some examples of how it is possible to tune the behavior of the spiking neurons by changing the values of the parameters in the neuronal model.

2.2 Transfer function

As mentioned in the previous chapter, the function that maps the input signal S to the average Post Synaptic Current (PSC) I that is perceived by the next (spiking) neuron is known as transfer function. Thus, even if in the present thesis a neural coding approach based on spike-time coding is used, the concept of transfer function stems from a rate-coding approach. Originally, in fact, based on a rate-code interpretation, the firing rate of the neuron was modeled with an activation function $f(S)$, which represented the frequency of the spikes along the axon, and the transfer function used for NNs was then the sigmoid function $f(S) = \frac{1}{1+e^{-S}}$, which is symmetric about 0.5 and has two horizontal asymptotes (in 1 and 0, respectively). However, since no negative activations are allowed by using the sigmoid, the hyperbolic tangent ($tanh$), i.e. a particular case of sigmoid function symmetric about zero, was considered. A very different variation of the sigmoid neuron is given by the Rectified Linear Unit (ReLU), expressed as $f(S) = \max(0, S)$, which, thanks to the convenient gradient, when chosen leads to easily trainable networks. In Figure 2.2 a tanh neuron and a ReLU neuron are displayed (compare with Figure 1.2 in the Introduction).

As mentioned in the Introduction, the training procedure of a NN consists of it-

eratively updating the synaptic weights by means of the backpropagation algorithm, during which the derivative of the transfer function is computed at the level of each individual neuron. For this reason, a network needs to be made of differentiable components in order to be trained through backpropagation, and a function which has derivative equal to 1 such as the ReLU makes it easier for the training procedure to update the weights. In the past, the use of ReLU was the solution to the vanishing gradient problem, for which the gradients assume increasingly smaller values, hampering thus the learning process. For the gating mechanisms typical of LSTM units, however, the ReLU is not a good candidate, because such a linear shape is not able to describe the gating behavior. For the gates, in fact, saturation (i.e. the presence of horizontal asymptotes) is necessary: a gate should ideally be either completely open (i.e. asymptote in 1) or completely closed (i.e. asymptote in 0).

For this project, the training procedure was chosen to be performed on analog neurons (because spike-based learning is slow and poorly effective), and then, once the training process ends (i.e. once the weights of the connections were set such as an error is minimized), the analog units were switched with spiking units while keeping the weights fixed, and the obtained spiking networks were used to perform the tasks. In order for this approach to be valid, the problem of finding an errorless superposition between the behavior of analog and spiking neurons has to be addressed. Here, it was decided to base this superposition on the neuronal transfer function, which translates into looking for a mathematical expression of the spiking transfer function. In fact, by applying such mathematical expression as transfer function for the analog neurons, the single units (analog and spiking) would process inputs in the same way, guaranteeing thus the possibility to train the analog networks off-line and take advantage of the state-of-the-art techniques in Deep Learning, being sure then when the analog nodes are switched with spiking nodes the network would solve the tasks it was trained to solve. In Figure 2.3 a schematic depiction of the analog/spiking conversion can be found.

An analytical transfer function describing the activation of spiking neurons was already exactly derived for the case $\tau_\beta = \tau_\eta = \tau_\gamma$ and approximated for the case $\tau_\beta = \tau_\eta \neq \tau_\gamma$ by considering a constant current input and equilibrium values for ϑ, $\hat{S}$, and I. For completeness the mathematical procedure leading to such functions is shown in the Appendix.

However, when testing the obtained transfer function on a simple network of three neurons (i.e. one input, a CEC, and one output) an offset between the update of the analog and the spiking CEC was noticed[57]. This error would lead to a mismatch between how the analog neurons are trained and the spiking neurons perform the tasks. Moreover, the results of the same study reported high firing rates which, together with some specific features of the LSTM architecture designed back then (which will be described later in this chapter), lead to, overall, low biological plausibility for the system. These are the reasons why it was decided to try the implementation of a stochastic adaptive spiking neuron in the first place.

Therefore, in order to obtain a better neuronal model, two new transfer functions are derived and tested in this project: an analytical solution, i.e. a variation from the original study, for a different ratio between τ_η and τ_γ (specifically, $\tau_\eta = 2\tau_\gamma$, which reflects more the relationship between the biological values of these time constants), and a transfer function describing the stochastic behavior of the neurons.

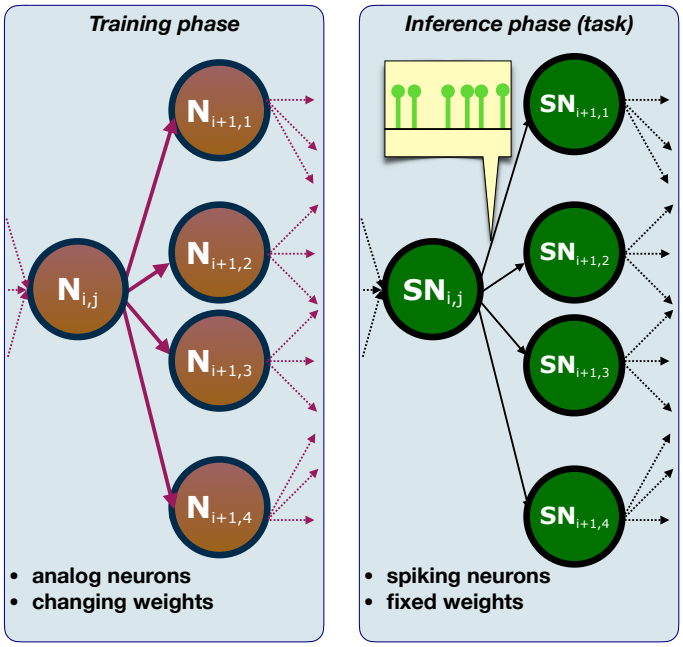

Figure 2.3: *Analog/spiking conversion. Left panel: training phase, the nodes are analog neurons and the weights are iteratively updated; right panel: inference phase, the analog nodes are switched with spiking units, the weights obtained at the end of the training process are now kept fixed.*

2.2.1 Analytical solution

As previously mentioned during the discussion of Figure 2.1, typical values for the time constants τ_η and τ_γ are not on a ratio 1:1. Specifically, the decay of the internal state over time is usually much slower than the decay of the threshold. For this reason, when wondering which other function can be tested as transfer function, it would be useful to have the analytical solution for a situation in which there is a different ratio between the two time constants.

In order to find the analytical solution for $\tau_\eta = 2\tau_\gamma$, the assumptions and the first part of the procedure shown in Appendix A are used, for which the transfer function $f(S)$ is given by the average value of $I(t)$. Thus, the following proceeding will have as a goal to express the average $I(t)$ as a function of the input S while including also the condition $\tau_\eta = 2\tau_\gamma$.

The starting point consists of noticing that, since the variables of the ASN decay exponentially (with time constants τ_β, τ_η, and τ_γ), they converge asymptotically. Under the assumption of a constant input, a steady-state situation can be considered. Hence, at any spike time t_f, I, ϑ and $\hat{S}$ reach always the same value:

$$I(t_f) =: I_f \,, \tag{2.6}$$
$$\vartheta(t_f) =: \vartheta_f \,, \tag{2.7}$$
$$\hat{S}(t_f) =: S_f \,, \tag{2.8}$$

before being incremented by an amount equal to h, $m_f\vartheta(t_f)$, and $\vartheta(t_f)$, respectively.

By setting $\tau_\beta = \tau_\eta$ (because of the reason expressed in Section 2.1) and $t = 0$ at the moment the last spike was fired, Equations 2.1, 2.3, and 2.4 can be rewritten, for

$0 < t < t_f$, as a function of these new quantities:

$$I(t) = (I_f + h)e^{-\frac{t}{\tau_\eta}}\,, \tag{2.9}$$

$$\vartheta(t) = \vartheta_0 + (\vartheta_f - \vartheta_0 + m_f\vartheta_f)e^{-\frac{t}{\tau_\gamma}}\,, \tag{2.10}$$

$$\hat{S}(t) = (\hat{S}_f + \vartheta_f)e^{-\frac{t}{\tau_\eta}}\,. \tag{2.11}$$

By assuming that the Inter-Spike Interval (ISI), which is the time distance between two consecutive spikes, is constant, the next time a spike is fired can be set as $t_s := \text{ISI}$, hence at $t = t_s$ the following conditions are obtained:

$$I(t_s) = I_f\,, \tag{2.12}$$

$$\vartheta(t_s) = \vartheta_f\,, \tag{2.13}$$

$$\hat{S}(t_s) = S_f\,, \tag{2.14}$$

and the condition of firing translates then into the limit $S - \hat{S}_f = 0.5 \cdot \vartheta_f$.

Firstly, an expression for the transfer function can be derived from Equation 2.9 and Equation 2.12:

$$\begin{aligned} f(S) &= \overline{I(t)}\Big|_0^{t_s} \\ &= \int_0^{t_s} (I_f + h)e^{-\frac{t}{\tau_\eta}}\,\mathrm{d}t \cdot \frac{1}{t_s} \\ &= -\frac{\tau_\eta}{t_s}(I_f + h)\left(e^{-\frac{t_s}{\tau_\eta}} - 1\right) \\ &= \frac{h \cdot \tau_\eta}{t_s}\,, \end{aligned} \tag{2.15}$$

thus, an expression for t_s as a function of S is needed. It can be obtained by solving the system built by putting together Equation 2.10 and 2.13, Equation 2.11 and 2.14, and the limit of the firing condition:

$$\begin{cases} \vartheta_0 + (\vartheta_f - \vartheta_0 + m_f\vartheta_f)e^{-\frac{t_s}{\tau_\gamma}} = \vartheta_f\,, \\ (\hat{S}_f + \vartheta_f)e^{-\frac{t_s}{\tau_\eta}} = \hat{S}_f\,, \\ S - \hat{S}_f = 0.5 \cdot \vartheta_f\,, \end{cases} \tag{2.16}$$

which can be solved for the limit quantities, giving:

$$\begin{cases} \vartheta_f = \vartheta_0 \dfrac{1 - e^{-\frac{t_s}{\tau_\gamma}}}{1 - (m_f + 1)e^{-\frac{t_s}{\tau_\gamma}}}\,, \\ \hat{S}_f = -\dfrac{\vartheta_f e^{-\frac{t_s}{\tau_\eta}}}{1 - e^{-\frac{t_s}{\tau_\eta}}}\,, \\ \hat{S}_f = S - 0.5 \cdot \vartheta_f\,. \end{cases} \tag{2.17}$$

Furthermore, by plugging the last equation of System 2.17 in the second last equation, and by solving for ϑ_f, it leads to the following equation:

$$\frac{2S(1 - e^{-\frac{t_s}{\tau_\eta}})}{1 - 2e^{-\frac{t_s}{\tau_\eta}}} = \vartheta_0 \frac{1 - e^{-\frac{t_s}{\tau_\gamma}}}{1 - (m_f + 1)e^{-\frac{t_s}{\tau_\gamma}}}\,, \tag{2.18}$$

which can be further solved, thus giving, eventually:

$$(2S+\vartheta_0)e^{-\frac{t_s}{\tau_\eta}}-[2S(m_f+1)+\vartheta_0]e^{-t_s\left(\frac{1}{\tau_\gamma}+\frac{1}{\tau_\eta}\right)}+[2S(m_f+1)-\vartheta_0]e^{-\frac{t_s}{\tau_\gamma}}=2S-\vartheta_0\,. \quad (2.19)$$

From here, the original approach assumed the condition $\tau_\gamma = \tau_\eta$, whilst for the present study the condition $\tau_\gamma = \tau_\eta/2$ is applied instead. By first making this substitution and then using $x = e^{-\frac{t_s}{\tau_\eta}}$, the following steps can be made:

$$\begin{aligned}
&(2S+\vartheta_0)e^{-\frac{t_s}{\tau_\eta}}-[2S(m_f+1)+\vartheta_0]e^{-3\frac{t_s}{\tau_\eta}}+[2S(m_f+1)-\vartheta_0]e^{-2\frac{t_s}{\tau_\eta}}=2S-\vartheta_0\\
&(2S+\vartheta_0)x-[2S(m_f+1)+\vartheta_0]x^3+[2S(m_f+1)-\vartheta_0]x^2=2S-\vartheta_0\\
&[2S(m_f+1)+\vartheta_0]x^3-[2S(m_f+1)+\vartheta_0]x^2+2\vartheta_0x^2-2\vartheta_0x-(2S-\vartheta_0)x\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad+2S-\vartheta_0=0\\
&[2S(m_f+1)+\vartheta_0]x^2(x-1)+2\vartheta_0x(x-1)-(2S-\vartheta_0)(x-1)=0\\
&[2S(m_f+1)+\vartheta_0]x^2+2\vartheta_0x-2S+\vartheta_0=0
\end{aligned}$$

which eventually lead to the solutions:

$$x = e^{-\frac{t_s}{\tau_\eta}} = \frac{-\vartheta_0 \pm \sqrt{4S^2-2S\vartheta_0 m_f}}{2S(m_f+1)+\vartheta_0}\,. \quad (2.20)$$

Of the two solutions, only the one with the plus sign is acceptable, since the other would give a negative argument for the logarithm. Thus, an expression for t_s is found to be

$$t_s = -\tau_\eta \ln\left[\frac{-\vartheta_0+\sqrt{4S^2-2S\vartheta_0 m_f}}{2S(m_f+1)+\vartheta_0}\right], \quad (2.21)$$

and, even if the intermediate steps are not reported here, it can then be plugged into the previously derived expression for the transfer function, i.e. Equation 2.15, obtaining:

$$f(S) = \frac{h\cdot\tau_\eta}{t_s} = \frac{-h}{\ln\left[\frac{-\vartheta_0+\sqrt{4S^2-2S\vartheta_0 m_f}}{2S(m_f+1)+\vartheta_0}\right]}, \quad (2.22)$$

in order to normalize the output of the analog neurons.

While h is kept equal to 1 in the spiking neurons, for the analog neurons it needs to be used as normalization factor, i.e. it can be set equal to the inverse of the limit for $S \to \infty$ of $f(S)$:

$$h = \frac{1}{\lim_{S\to\infty} f(S)} = \frac{1}{\frac{1}{\ln\frac{1}{m_f+1}}} = \ln\frac{1}{m_f+1}\,. \quad (2.23)$$

The newly found function and its derivative were tested, as described in the next chapter, by means of the same three-neuron network originally used, in order to observe if this more plausible ratio between τ_η and τ_γ can solve the issue of mismatching CECs.

2.2.2 Stochastic neuron

As previously anticipated, another transfer function was studied as well. Specifically, a function describing the behavior of a stochastic ASN, which is thus more biologically plausible and expected to be computationally convenient. In this case, since the mathematical derivation of the function was not possible because of the instability of the signal, it was chosen to simulate how a stochastic ASN would behave when injected with a constant input, and to study its output. Here, a description of the stochastic ASN is provided.

Starting from a stochastic model for the spiking neuron, it was assumed that the ASN emits spikes following a stochastic firing condition defined as:

$$\lambda(V(t), \vartheta(t)) = \lambda_0 \exp\left(\frac{V(t) - \vartheta(t)/2}{\Delta V}\right), \tag{2.24}$$

where $V(t)$ is the membrane potential defined as the difference between $S(t)$ and $\hat{S}(t)$, λ_0 is a normalization parameter and ΔV is a scaling factor that defines the slope of the stochastic area[51]. Biological neurons are, in fact, not deterministic, and the stochasticity would permits the description of spike firing also below threshold, giving thus a continuous–hence more convenient–transfer function.

In this case, the transfer function was not derived mathematically, but approximated by fitting the activation function obtained from simulations of the activity of stochastic neurons in Matlab. The fitting procedure is described in the next chapter. The function used was the following:

$$f(S) = \frac{1}{a\exp(bS) + c\exp(dS) + 1}. \tag{2.25}$$

The approximation of the stochastic transfer function, once optimized, will be used for the experiments involving the Machine Learning tasks presented in the next chapter.

The adaptive, stochastic, spiking LSTM

As introduced in the previous chapter, the LSTM unit is an architecture which allows to introduce the time in the neural net, making it of the recurrent type. With the exception given by the TrueNorth study mentioned in the Introduction, no spiking LSTMs were developed so far.

For the part of this project involving the testing of SNNs by means of Machine Learning tasks, an analog LSTM was present in the network during the training phase, and then, in the same way as the rest of the units in the networks, substituted with a spiking counterpart. The unit used here is stripped of many (optional, for the tasks involved in this project) features, such as forget and output gates, or recurrent weights. Moreover, while the original approach (i.e. the one using the analytical transfer function with $\tau_\eta = \tau_\gamma$) used three neurons in order to get the desired transfer function in both the input gate and the input cell of the spiking LSTM (see [56]), here a single neuron is used in both cases, obtaining thus a much more simplified unit. A picture of the old structure is showed in the Appendix.

Overall, the analog LSTM in this project uses a multiplicative input gate, an input cell, a CEC, and an output cell, and the spiking one, instead, is composed only of a

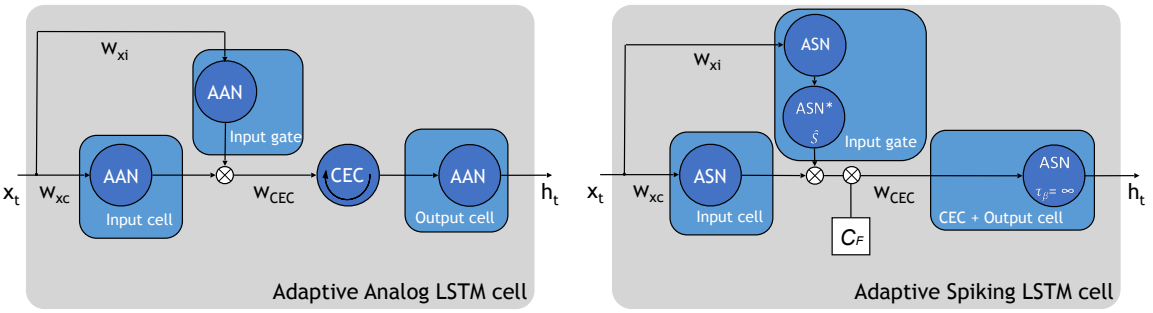

Figure 2.4: *Overview of the simplified construction of an Adaptive Analog LSTM (left) and an Adaptive Spiking LSTM cell (compare to the architecture showed in the Appendix). This compares to an LSTM with only an input gate.*

multiplicative input gate, an input cell, and an output cell. In fact, the spiking CEC was programmed like a normal spiking neuron, but without the decay of the internal state $\hat{S}(t)$, for which the time constant was theoretically $\tau_\beta = \infty$, i.e. the exponential was set equal to 1, since it is possible to make the assumption that this decay is very slow for these cells[58]. In Figure 2.4 a depiction of the analog and spiking LSTM is shown: W_{xc}, W_{xi}, and W_{CEC} are the weights for the input received by the input cell, the input gate, and the CEC, respectively, while the ASN* neuron is an intermediate neuron accumulating the signal coming from the input gate, and its internal state $\hat{S}$ is multiplied with the signal going from the input cell to the output cell.

Concerning the update of the CEC, if the ISI can be considered constant the output is increased by $\frac{h}{\text{ISI}}$ every time step. In the case of a stochastic unit, the mean ISI (i.e. $\overline{\text{ISI}}$) can be considered constant. Thus, the analog CEC should be updated by $\frac{h}{\overline{\text{ISI}}} = \frac{S}{\tau_\eta}$ (see Equation 2.15, for which the derivation still holds, by assuming $t_s := \overline{\text{ISI}}$ from the start instead of $t_s := \text{ISI}$). However, in order to simply update the analog CEC by S, the spiking CEC is instead updated at every time step as follows:

$$\text{CECstate}(t) = \text{CECstate}(t-1) + S \cdot \text{C}_F\,, \tag{2.26}$$

where S is the input current entering the CEC. The parameter $\text{C}_F = \tau_\eta/\Delta t$ is a conversion factor, with Δt being the time of the spiking simulation, and it needs to be multiplied to the incoming signal entering the spiking CEC in order to have an accurate conversion from the analog CEC. The analysis of the time component in the two networks represents a crucial point for understanding how to correctly perform the switching from analog to spiking neurons. More in detail, while in the analog environment the input is presented once to the network, in the spiking networks it is required to show the input for several time steps, i.e. milliseconds, in order for the spiking neurons to pass the refractoriness stage (i.e. the phenomenon described in the Introduction for which the neuron needs time in order to adapt to the incoming signal) and to reach a point at which the communicated signal is (to some extent) stable. For this reason, the spiking CEC would store more information than the information an analog CEC would have accumulated. Hence the Δt at the denominator in the C_F for the update of the spiking CEC.

2.3 Machine Learning paradigms

In order to test a NN different Machine Learning tasks can be used. In this section the theory of learning is presented, even if it is worth noticing that the focus of this thesis is the optimization of the AAN-ASN conversion.

Overall, it is possible to distinguish between three different types of algorithms, namely supervised learning, unsupervised learning, and reinforcement learning tasks. These terms, to be precise, are not simply algorithms, but they refer both to problems, solution methods, and fields that study those problems and solution methods. Since unsupervised learning algorithms are used for problems such as clustering (their common goal is, in fact, to find hidden structures in sets of unlabeled data), in the present work only the first and the third categories are considered, since they are more related to the type of problems an LSTM network can be deployed for.

Specifically, supervised learning problems refer to situations in which the training consists of comparing the expected output with the output actually produced by the network, and iteratively reducing the residual between these two quantities. The goal of these learning algorithms is to make a system extrapolate its responses so that it will be able to correctly behave in future (unknown) situations. A typical example is handwriting recognition, a widely used supervised learning task for which handwritten figures are shown to the network together with the correct labels for each digit in order for it to find what correlates the input and the expected outputs. This type of tasks are very straightforward and easy to implement, but they require a known set of correct input-output pairs.

On the other hand, during reinforcement learning tasks such a requirement is avoided. In fact, in this case the learning is based on rewarding mechanisms for which an agent tries different possibilities and it is rewarded whenever something is well done, i.e. the same type of process through which biological NNs are supposed to learn[59]. A reinforcement learning system is composed by: an agent, an environment, a policy, a reward signal, a value function, and an (optional) model describing the environment. Reinforcement learning agents have specific goals, can interact with the environment, and can choose actions to take in order to influence it, and they do so while compromising between exploration and exploitation: by exploiting what was already learned, a sensible choice can be already made, but by making more exploration better actions could be made in the future. The policy is a function that maps a state into an action to take, and it is what determines the behavior of the agent. The reward signal is what sets the goal for the agent: each action receives feedback in terms of a reward, which gives to the agent a sense of whether the action that was just taken was good or not, and its final objective is to maximize the total reward receivable. Another indication about which action to choose is given by the value function: it indicates the long-term desirability of an action by giving, in each state, the amount of reward obtainable in the entire run for each possible action that can be taken at that point. Finally, models are used for planning: sometimes it is possible, in fact, to give some a priori knowledge about how the environment will behave.

In the end, it must be mentioned that while deep NNs are widely used for instructive feedback learning (i.e. supervised learning), in order to perform reinforcement learning tasks Deep Learning is not necessary. Other solution methods not involv-

ing deep nets exist, in fact. However, when the sizes of the states and the policy increase, it could be convenient to rely on NNs in order to replace a (quickly intractable) table-like policy with more convenient functions providing the greediest actions over time.

Chapter 3

Experiments

As explained earlier, the first part of this project involved the mathematical derivation or approximation of new transfer functions, namely an analytical solution and a stochastic one, for the analog neurons. First, the analytical solution with more biologically plausible time constants was considered and tested by comparing the update of the analog and spiking CECs. Then, the stochastic solution was derived by means of a fitting procedure, and tested as well. Subsequently, the same stochastic solution was implemented in a LSTM network, with which a Sequence Prediction task and a T-Maze task were trained. For each experiment, the AANs and ASNs described in the previous chapter were used.

3.1 Transfer function

In this section, at first, the experimental setup (i.e. the values for the parameters of the neuronal model) used for testing the analytical transfer function is described. Subsequently, the process leading to the finding of the stochastic solution is presented, followed by, again, the description of the setup used.

3.1.1 Analytical transfer function

The transfer function found in Section 2.2.1 was implemented in Python. The code comprised of a two three-neuron architectures (i.e. one analog and the other spiking), in which the first neuron gave the input, the second was a CEC, and the third neuron acted as a readout. In Figure 3.1 a depiction of the three-neuron system is presented, for both the analog and the spiking case. The simulations were run twice. At first, in both networks the time constants of the neurons were $\tau_\eta = 30$ ms, $\tau_\gamma = 15$ ms, while for the second simulation they were changed to $\tau_\eta = 20$ ms, $\tau_\gamma = 10$ ms. For both the simulations, the multiplicative factor and the initial threshold were $m_f = \vartheta_0 = 0.1$. Concerning the AANs, the input an readout units were set to have the analytical transfer function (Equation 2.22) with $h = \ln\left(\frac{1}{m_f+1}\right)$ (i.e. the limit for $S \to \infty$ of the transfer function, as shown in Equation 2.23), while the state of the CEC was initialized to zero and then updated by means of the expression in Equation 2.26.

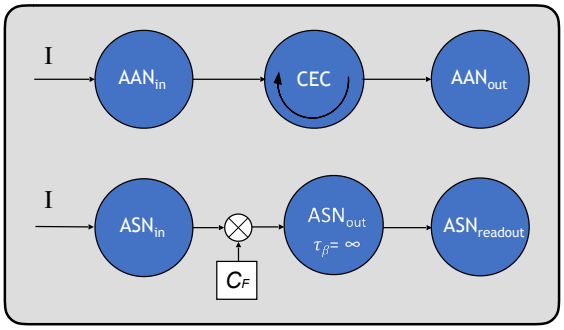

Figure 3.1: *Three-neuron experiment: architecture implemented in Python and used to test the transfer functions during this project. The top row of neurons represent the analog system, for which an input neuron, a CEC, and an output neuron were deployed; the bottom line of neurons consists of three spiking neurons: an input neuron, a neuron with $\tau_\beta = \infty$ (i.e. a CEC), and a readout neuron.*

The ASNs, instead, were characterized by having the firing condition in Equation 2.5 and the time constant $\tau_\phi = 2.5$ ms.

Both the analog and the spiking networks were studied for 200 time steps (milliseconds, in the spiking simulation), with an input current $I = 1.0$ fed into the system from the 20th till the 180th step. For the purpose of comparing the results, the same simulations were performed for the original transfer function, at first with $\tau_\eta = \tau_\gamma = 10$ ms and then with $\tau_\eta = \tau_\gamma = 10$ ms.

3.1.2 Stochastic transfer function

In order to find a transfer function for the stochastic AANs able to describe the way an adaptive stochastic neuron works, a code providing the evolution of such a system was written in Matlab. This code depicted an ASN to which a constant input current was fed, and for which the condition of firing was the following:

$$\begin{cases} S - \hat{S} > 0\,, \\ P = N \exp\left(\dfrac{S(t) - \hat{S}(t) - \vartheta(t)/2}{\Delta V}\right)\,, \\ X < P\,, \end{cases} \tag{3.1}$$

where the first equation is needed in order for the neuron to avoid having responses for negative signals (which are not approximated in the case under investigation because not biologically plausible and because it would subvert the conversion between analog and digital signals), while the second equation shows the probability of firing (equivalent to the expression in Equation 2.24), and the third expression represents the new condition of firing. In synthesis, at every step the difference $S - \hat{S}$ is evaluated, if it is greater than zero a probability P is calculated and a random number X is extracted and they are compared: if X (drawn randomly at each step by means of the built-in function `rand`) is smaller than the probability P the neuron fires a spike, giving thus a system describing a purely stochastic neuron.

For different values of the input current I the output of the above-described neuron was studied over a time span Δt and the mean of the output signal recorded over time was calculated.

Tuning the parameters

Before being able to perform the simulation leading to the optimal stochastic transfer function, some parameters needed to be optimized. In detail, the normalization factor N and the steepness ΔV were tuned at first, in order for the description of the firing probability to fulfill some criteria, and subsequently the simulation time Δt was varied and the resulting transfer functions with related errors were analyzed.

By implementing in Matlab the stochastic neuron with the characterization described in these chapters, the output can be studied in order to choose the values for these parameters which give the optimal transfer function. The ideal transfer function has a sigmoid-like shape, i.e. with two asymptotes one of which needs to be at zero, it reaches the upper asymptote relatively soon (i.e. at low values of input current) with a not steep slope. Also, the values obtained for P have to lie in the range of values between zero and one (i.e. it needed to be normalized). Such a function is particularly needed because of the gating mechanism implemented in the LSTM unit: the gate needs to have a continuous transfer function going from 0 (i.e. closed gate) to 1 (i.e. open gate).

Different combinations of values for ΔV and N were tried, before settling for a certain shape for the transfer function. At first, N was kept fixed at an arbitrarily-chosen value of 0.01 while letting ΔV vary, and then, once the best value for ΔV had been identified, it was kept fixed while N was changed. For these simulations, a total simulation time of 500 ms was set, since that amount of time was enough for the neuron to reach a stable signal, for values of current from 0 to 4.8, with steps width equal to 0.05; concerning the neuronal parameters, the following values were chosen: $\tau_\eta = \tau_\gamma = 10$ ms, $\vartheta_0 = 0.3$, $\tau_\phi = 2.5$ ms, and $m_f = 0.6 \cdot \vartheta_0$. These values were chosen for giving a low firing rate but while still keeping fast adaptation.

Then, once the two stochastic parameters were fixed, different plots were made while changing the simulation time (namely, with $\Delta t = [50, 100, 150, 200]$ ms, and they were compared with the transfer function obtained at very long simulation time, i.e. $\Delta t = 500$ ms like before, in order to find a value of Δt that allows to obtain an accurate transfer function without requiring too much time for the neurons to pass the refractory period. For this experiment, the same neuronal parameters as before were used.

Fitting procedure and testing

Once the optimal values for the simulation parameters were chosen, the final stochastic neural model could be implemented in the Matlab code simulating the stochastic ASN, and the transfer function obtained could be plotted and fitted.

Before the fitting procedure, however, the obtained spiking activation function was normalized at the point where it reached a plateau, in order to set the upper asymptote at approximately one. Then, the normalized spiking activation function was fitted with a sum-of-exponentials shaped function as seen in Equation 2.25. The fitting process was performed by means of Matlab and the Curve Fitting Toolbox software.

The final transfer function, similarly as for the analytical solution, was at first tested in a simple three-neuron code, for which the first and the third neurons served as input and output unit, respectively, and the neuron in between was coded as

a CEC. In the analog system, the two neurons were given the stochastic transfer function, and the CEC was updated as in Equation 2.26. Differently than in the other experiments on the three-neuron architecture, in this case the analog network was studied for 10 time steps, each of which corresponded to 200 ms in the spiking simulations (i.e. the total duration was 2000 ms).

3.2 Implementation and tasks

After a transfer function coinciding with the spiking model has been determined, networks made of such units can be tested. In particular, verifying how a spiking LSTM network would perform in training- and memory-requiring tasks was the main focus of this thesis.

For this project, in order to demonstrate the generality of the approach, four different tasks were performed: two versions (with and without noise) of Sequence Prediction, i.e. an unsupervised learning task, and of T-Maze, i.e. a reinforcement learning problem, respectively.

3.2.1 Sequence Prediction

The main concept of LSTM, the ability of a CEC to maintain information over long stretches of time, was demonstrated in [9] in a Sequence Prediction task: the network has to predict the next input of a sequence of $p+1$ possible input symbols denoted as $a_1, ..., a_{p-1}, a_p = x, a_{p+1} = y$. In the *noise free* version of this task, every symbol is represented by the $p+1$ input units with the $i-th$ unit set to 1 and all the others to 0. At every time step a new input of the sequence is presented. As in the original formulation, the network here is trained with two possible sequences, $(x, a_1, a_2, ..., a_{p-1}, x)$ and $(y, a_1, a_2, ..., a_{p-1}, y)$, chosen with equal probability. For both sequences the network has to store a representation of the first element in the memory cell for the entire length of the sequence p. A schematic depiction of the noiseless Sequence Prediction task is presented in Figure 3.2.

In the *noisy* version of the Sequence Prediction task, the network still has to predict the next input of the sequence, but the symbols from a_1 to a_{p-1} are presented in random order and the same symbol can occur multiple times. Therefore, only the final symbols a_p and a_{p+1} can be correctly predicted. This version of the task avoids the possibility that the network learns local regularities in the input stream.

For the noiseless task 20 networks were trained for a total of 100k trials, with $p = 100$, while the same number of networks were trained in the noisy version but for 200k trials. For every trial in both tasks the network comprised of $p+1$ input units and $p+1$ output units. The input units were fully connected to the output units without a hidden layer. The same sequential network construction method from the original paper was used to prevent the *abuse problem*, for which the weights of the LSTM unit are pushed away from zero from the start (i.e. the part of the task which does not require memory) in an attempt to reduce the error. In practice, this translates to including the Adaptive Analog LSTM cell in the network only after the error stops decreasing[9]. The cost function for the training process in both versions

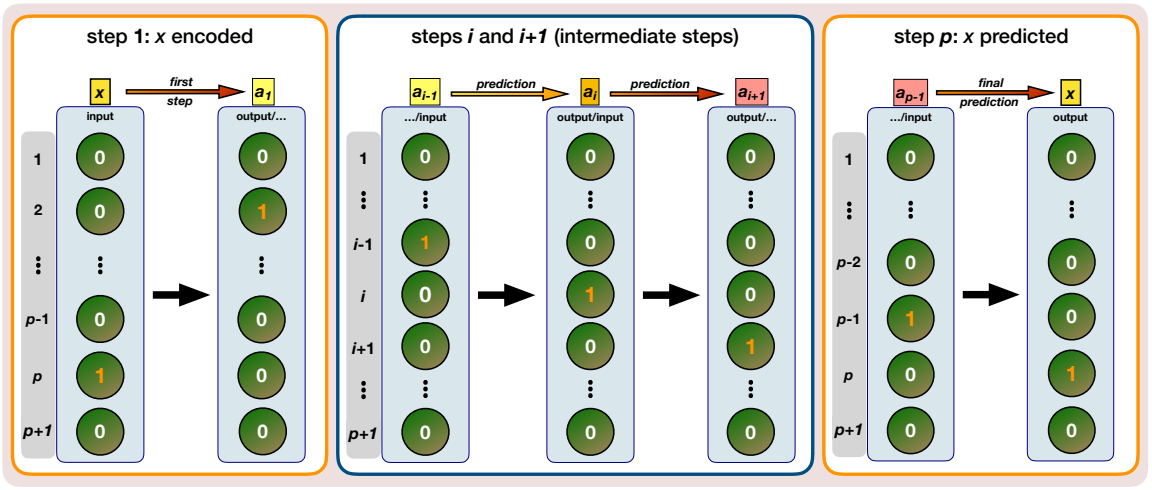

Figure 3.2: *Noiseless Sequence Prediction task for the* $(x, a_1, a_2, ..., a_{p-1}, x)$ *sequence. Left panel: first step, in which the first element of the sequence (which has to be kept in memory for the entire duration of the task) is presented; middle panel: intermediate steps, the position of the value 1 shifts along the vector at each step; right panel: final step, the network has to recognize that this is the last step and it has to give as prediction the vector initially presented (i.e. in this case the vector* x*).*

of the task was the cross-entropy error:

$$O = -\sum_{(x,z)\in S}\sum_{i=1}^{p+1} z_i \ln y_i + (1 - z_i)\ln(1 - y_i)\,, \tag{3.2}$$

with x being the input of the network, y_i and z_i the prediction and the target, respectively, of the i-th output neuron. This choice was made in order to have better performance in the training. For both noise-free and noisy tasks the network was considered having converged when the average error over the last 100 trials was less than 0.25.

Concerning the update rules during the learning procedure, a customized truncated version of Real-Time Recurrent Learning (RTRL) was used for the LSTM unit[60]. The regular analog neurons were trained by means of the backpropagation algorithm.

3.2.2 T-Maze

In the T-Maze task, originally introduced for an LSTM unit in [61], an agent has to move inside a maze to reach a target position in order to be rewarded while maintaining information during the trial. A different value, known as *Q-value*, is assigned to each possible action the agent can take (i.e. moving in four different directions), representing the utility of the action (learned during the learning process), and thus determining the probability of the action choice at each step. This reinforcement learning technique involving the learning of an action-value function is known as Advantage Learning (i.e. a modified version of Q-learning which operates in continuous time[62]), and one of its strengths is that it allows to compare the expected utility of the available actions without requiring a model of the environment.

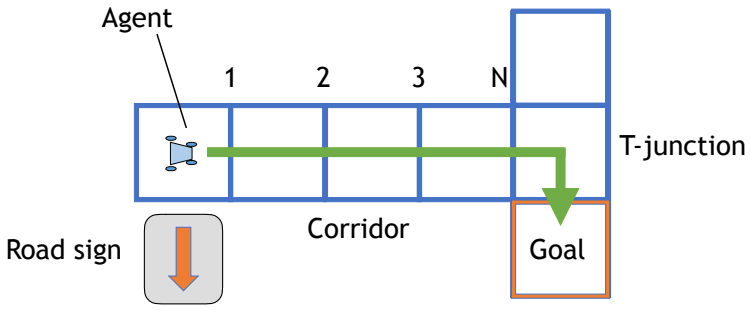

Figure 3.3: *T-Maze task with corridor length N. An agent receives an input ("road sign") at the beginning of its journey and it has to end up in the correct side ("goal") of the T-junction at the end of the simulation.*

The maze in a T-Maze task is composed of a long corridor with a T-junction at the end, where the agent has to make a choice based on information presented at the start of the task. In Figure 3.3 a schematic depiction of a T-Maze is shown.

In the present study, the corridor length was set to a value $N = 20$, the agent received a reward of 4 if it reached the target position and -0.4 if it moved against the wall. If it moved to the wrong direction at the T-junction it also received a reward of -0.4 and the system was reset. The negative reward value is chosen bigger in absolute value than the one originally used in [61] in order to encourage Q-values to differentiate more during the trial. The agent had three inputs and four outputs, corresponding to the four possible directions it could move to, and a fully connected hidden layer with twelve neurons and three Adaptive LSTMs was included. At the beginning of the task the input could be either 011 or 110, referring to the two possible sides of the T-junction at which the reward could be placed. Other inputs indicated the corridor and the T-junction: for the former, the input 101 was used for the noiseless version and $a0b$ for the noisy version (with a and b uniformly distributed random variables in a range of $[0, 1]$), whilst for the latter the input was 010 for both the noiseless and the noisy tasks. This setup was chosen as in [61] in order to compare the results.

The same training parameters as in [61] were used; again, 20 networks for each task were trained. As a convergence threshold, in the last 100 trials the network had to have reached on average a total reward greater than 3.5. The Max-Boltzmann controller[63] was used as exploration, i.e. action choice, mechanism. It selects the action with the highest Q-value, i.e. the greedy action, with probability $1 - \epsilon$ and a random action k with probability $\epsilon = 0.05$ sampled from the Boltzmann distribution:

$$P_B(k) = \frac{\exp(Q_k)}{\sum_{k'} \exp(Q_{k'})} . \tag{3.3}$$

For this task, the regular neurons were trained by means of traditional backpropagation, whilst for the memory unit RL-LSTM [61] was used as learning algorithm, which consists of the same as RTRL but preferable in the case of reinforcement learning methods, and particularly suitable to the practice of advantage learning[62], which, again, was used here in order to compare the outcome of the experiments with the results obtained originally by Bakker in 2002[61].

Chapter 4

Results

In this chapter, the main results from the experiments introduced previously are presented in the following order: at first, the comparison between the update of analog and spiking CEC during the three-neuron simulation is presented for the analytical transfer function derived in Chapter 2; subsequently, the tuning procedure concerning the parameters governing the stochastic behavior of the neuron is exploited, then the results of the same study about the update of the CECs as before are shown; the second part of the chapter, instead, focuses on the results obtained with the implementation of the stochastic transfer function for the solution of Sequence Prediction and T-Maze problems.

4.1 Analytical transfer function

As explained before, a neuronal model with $\tau_\eta = 2\tau_\gamma$ is a more biologically plausible system. The transfer function mathematically derived in Chapter 2 was tested, as described in the previous chapter, in a three-neuron system. This simple experiment allowed to check whether the analog and the spiking CECs were updating at the same pace, condition that was not verified in the previous study. The reason for this offset lies in the transient phase problem mentioned in Chapter 2, for which at the beginning of the simulations the firing rate is particularly high, causing a bump in the spiking

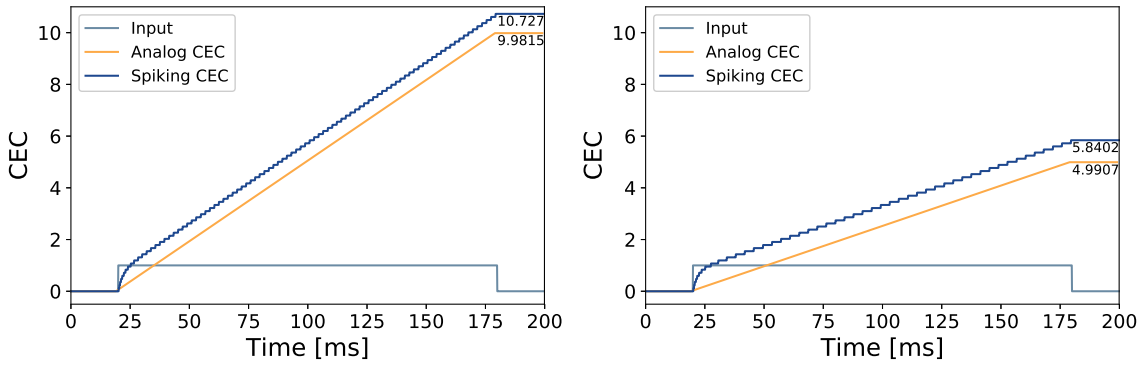

Figure 4.1: *Update of the analog and spiking CECs when the old transfer function is used: (left panel) $\tau_\eta = \tau_\gamma = 10$ ms, (right panel) $\tau_\eta = \tau_\gamma = 20$ ms.*

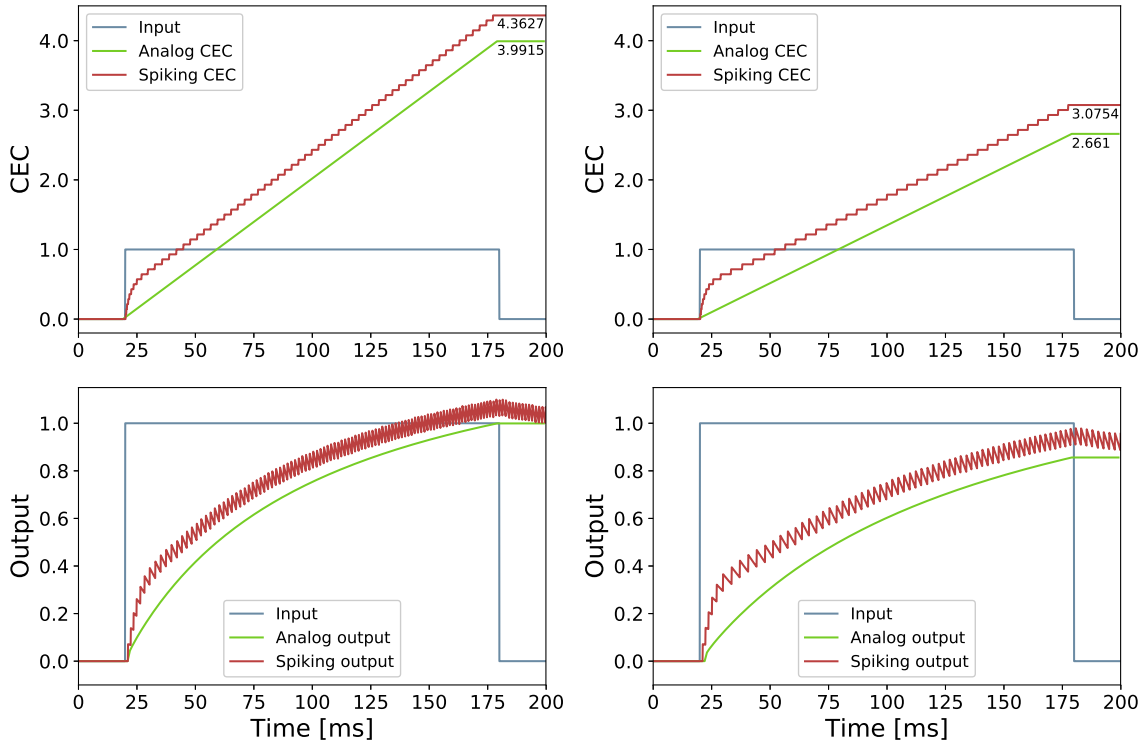

Figure 4.2: *Update of the analog and spiking CECs (top row) and output signal (bottom row) when the newly-derived transfer function is implemented: (left panel)* $\tau_\eta = 20$ *ms,* $\tau_\gamma = 10$ *ms; (right panel)* $\tau_\eta = 30$ *ms,* $\tau_\gamma = 15$ *ms.*

CEC plot. For comparison purposes, in Figure 4.1 are shown two plots underlying the just-mentioned problem, when the original transfer function was deployed (i.e. with $\tau_\eta = \tau_\gamma$), and made by means of the same three-neuron architecture that was previously described. The errors were 7.47% and 17.02%, respectively, whilst the firing rate for a single neuron was 450 Hz in the first experiment and 245 Hz in the second experiment.

In Figure 4.2, instead, in the top row of plots the evolution over time of the analog and the spiking CEC states is plotted. It was obtaining by running the previously described three-neuron system, but this time the newly derived transfer function was implemented. At first, the time constants were $\tau_\eta = 20$ ms, $\tau_\gamma = 10$ ms, while for the second simulation they were changed to the values $\tau_\eta = 30$ ms, $\tau_\gamma = 15$ ms. In this case, also the output of the CEC was plotted over time. In the analog system, it refers to the output of the output neuron, whilst in the spiking network this corresponds to the input received and filtered by the readout neuron. Even in this case, the analog and spiking curves do not match. It is noticeable that the problem of mismatching CEC states over time was not solved, but thanks to the new transfer function, lower firing rates are obtained (namely, they were equal to 320 Hz and 215 Hz respectively), even in the case of fast decays. This was due to the choice of the combination of values for the time constants, in accordance with what could have been expected (see Figure 2.1). The error in this case was of 9.30% and 15.57%, respectively, i.e. bigger than the original. In particular, $\tau_\eta = \tau_\gamma = 10$ ms gives better results probably because it has faster adaptation thus shorter transient phase. Since the results were not as good as hoped, the new analytical transfer function was not further tested.

4.2 Stochastic transfer function

As explained in the previous chapter, in order to find the stochastic transfer function the parameters ΔV and N in Equation 3.1 needed to be tuned at first. In Figure 4.3 (left panel) a plot showing how the (mean) probability (calculated over 500-ms-long simulations) changed as a function of the input current when the value of the parameter ΔV was changed, whilst keeping the value of N constant (an arbitrary value of 0.01 was chosen). The ideal trend for the probability approaches zero when

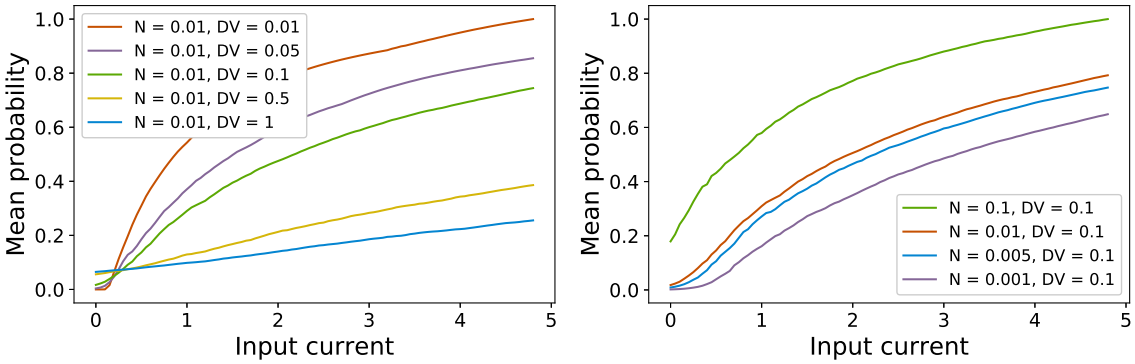

Figure 4.3: *Tuning the parameters controlling the firing probability of the neuron: (left panel) the scaling factor ΔV is varied while the normalization factor N is kept fixed at an arbitrary value, (right panel) the value chosen for ΔV is kept fixed while N is changed.*

the value of the current is close to zero and, for increasing values of current, it should increase as well and smoothly go towards saturation (at 1). These characteristics are desirable for the implementation of the transfer function in a gating mechanism. Hence, given these criteria, $\Delta V = 0.1$ (i.e. the green curve) seemed like the best compromise. Then, this value was used in other simulations in order to tune the value of N as well. The results can be seen in the right panel of Figure 4.3. For the same reasons as before, the value $N = 0.005$ (i.e. the blue curve) was eventually chosen. Once the values for ΔV and N were set, before the simulations for finding

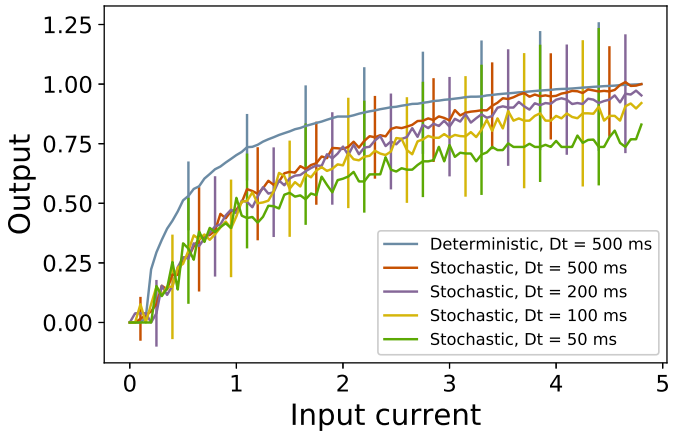

Figure 4.4: *Choice of the simulation time Δt. The functions are normalized to 1 with respect to the $\Delta t = 500$ ms functions.*

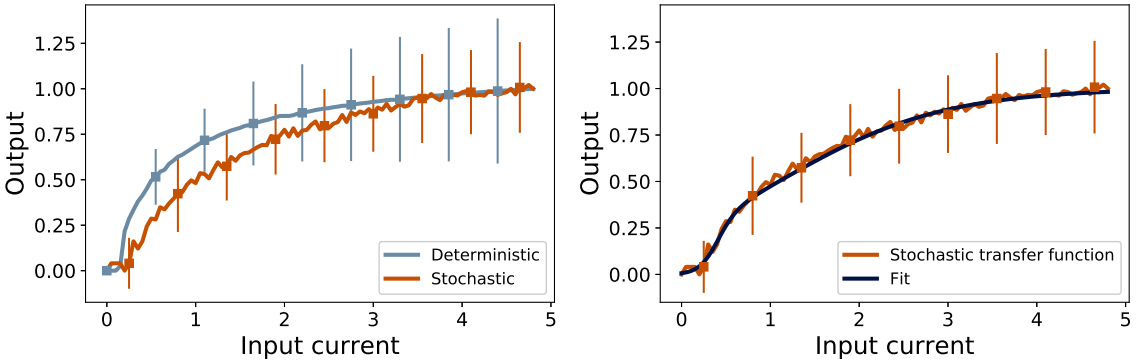

Figure 4.5: *Plots of the stochastic transfer function: (left panel) stochastic and deterministic data points plotted with their errorbars, (right panel) stochastic data with error and related fitting function.*

the transfer function could be performed, the study concerning the simulation time needed to be done. In Figure 4.4 the stochastic transfer functions obtained with $\Delta t = [50, 100, 200, 500]$ ms respectively are plotted as a function of the input current. The value $\Delta t = 200$ ms was selected to be implemented in the spiking LSTM because the related curve is close to the $\Delta t = 500$ ms curve and the errors look small enough, while also reducing a lot the simulation time. This is important because when solving Machine Learning tasks, for instance, if the simulation time for which the transfer function was obtained is reduced by a factor n, the total amount of time the task will require (for either training or testing) is reduced by the same factor.

Then, the simulations for the stochastic neuron could finally be made. In the left panel in Figure 4.5 the final transfer function obtained by means of the Matlab simulation is plotted, together with the deterministic function. The data points of the stochastic function were then fitted by means of the parametric function in Equation 2.25, with parameters:

$$\begin{aligned} a &= +\,148.7 \pm 23.0\,, \\ b &= -\,10.16 \pm 0.40\,, \\ c &= +\,3.256 \pm 0.049\,, \\ d &= -\,1.08 \pm 0.008\,. \end{aligned} \tag{4.1}$$

The goodness of the fit reported was in terms of the R-square (which was 0.999, i.e. almost all of the variance is accounted for by the fit). The fitting function can be seen plotted together with the relative data points of the stochastic transfer function in the right panel of Figure 4.5.

When tested in the three-neuron architecture, this function gave the results depicted in Figure 4.6. It can be noticed that the CECs were updating at the same pace (compare with Figure 4.1 and Figure 4.2, for which the spiking CEC state had a fast starting increase, leading to the issue of mismatching CECs). In order to compare these results with the previous ones, the value of the analog CEC in the first step was compared to the value of the spiking CEC after 200 ms, giving an error of 2%. Moreover, the firing rate was 45.5 Hz, i.e. much lower than before.

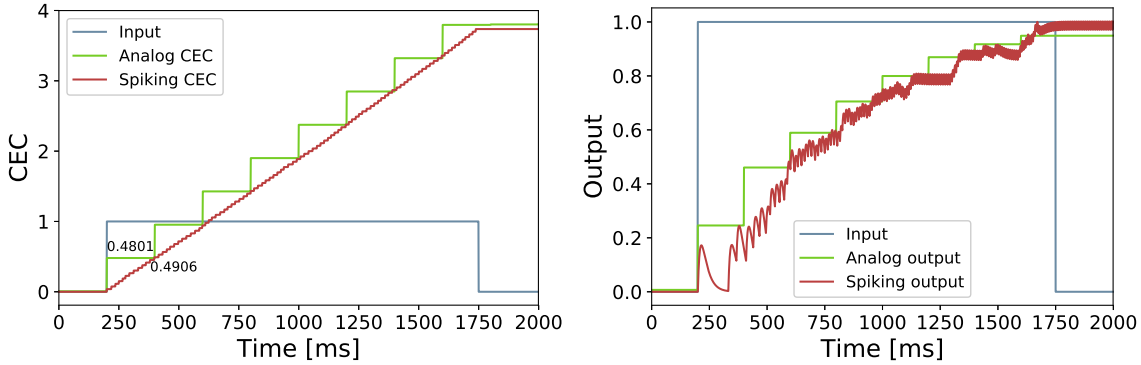

Figure 4.6: *Results from the three-neuron experiment: (left panel) update of the analog and spiking CECs during the simulation, (right panel) input signal received by the analog and spiking output neurons during the simulation. The values of the CECs after 1 step/200 ms since the input was injected are reported in the plot.*

4.3 Tasks

In Figure 4.7 the results from one of the networks before (left) and after (right) the conversion for the Sequence Prediction task are plotted. In particular, the last 6 time steps ($6 \cdot 200$ ms for the spiking simulations) are shown. The first row represents the task for which the y symbol had to be predicted (brown curve), whilst the bottom row of plots refers to the data collected when the network had to remember and predict the x symbol (pink curve). Each color refers to the output value of a specific neuron in the output layer, in particular the brown color shows the output over time of the second-last (the p-th) output neuron and the pink color does the same but for the last (the $(p+1)$-th) output neuron. In both cases, it can be noticed that the network was able to predict the correct output by the end of the simulations.

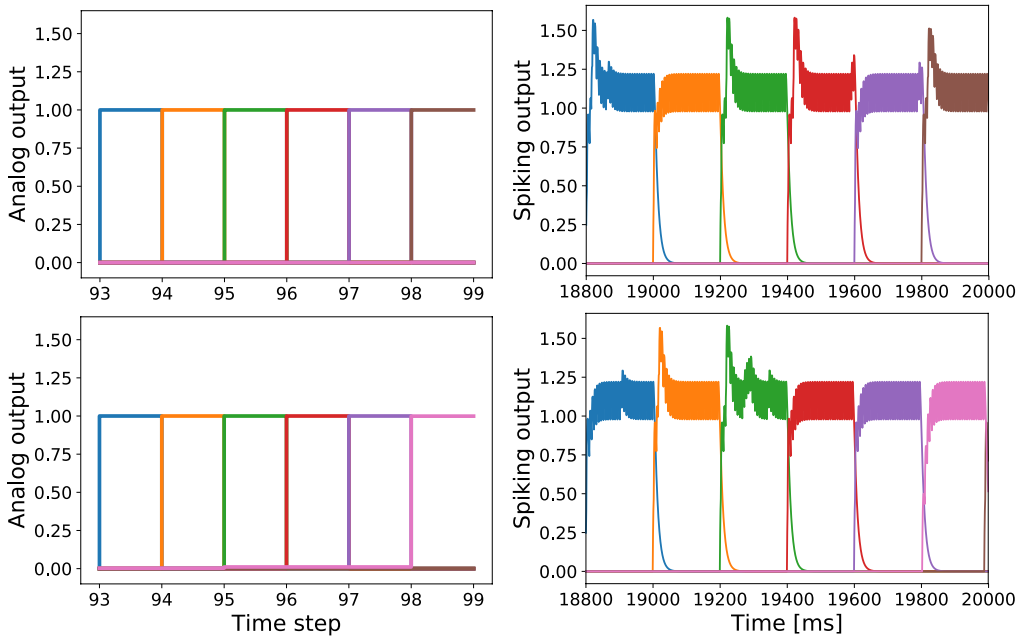

Figure 4.7: *Noiseless Sequence Prediction results. Top panels: x-symbol encoding; bottom panels: y-symbol encoding.*

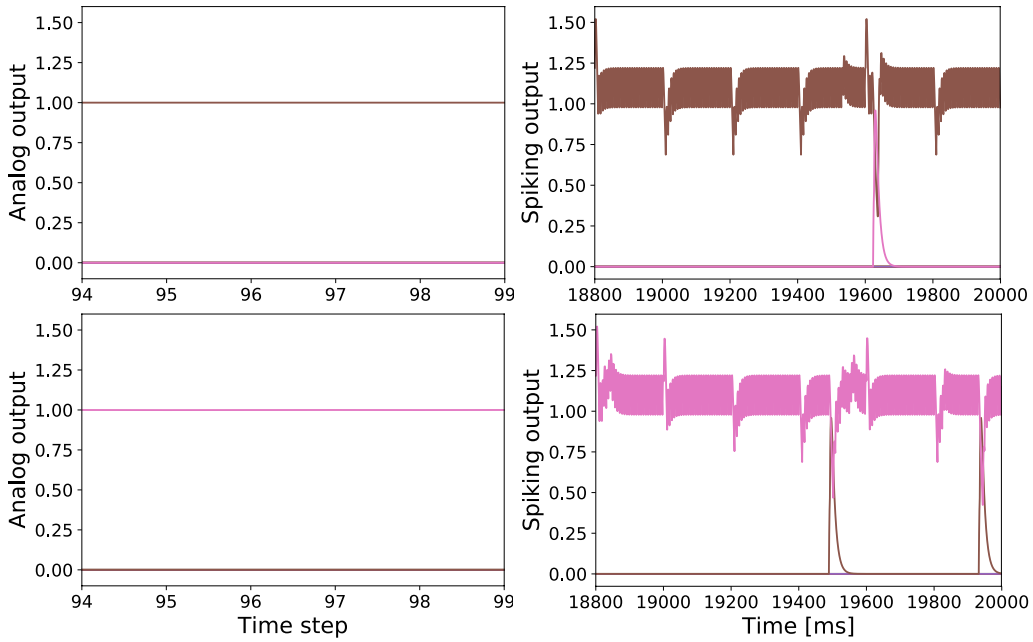

Figure 4.8: *Noisy Sequence Prediction results. Top panels: x-symbol encoding; bottom panels: y-symbol encoding.*

A similar collection of plots for the noisy Sequence Prediction task are shown in Figure 4.8. Again, it can be seen that the spiking network was able to successfully solve the tasks. The stochastic nature of the model causes some instabilities in the output (in the right-hand plots the brown and the pink curves intersect each other.

In Figure 4.9 the update of the state of the analog and spiking CECs over time are plotted for both the noiseless (left panel) and the noisy task (right panel). The CECs stored the most information at the beginning of the task, i.e. when the first input was given to the networks. More study will be needed in order to understand why the CECs behave in this way.

In Figure 4.10, instead, the results from one of the networks before (left) and after (right) the conversion for the noiseless T-Maze task are shown, consisting of plots of the evolution over time of the four Q-values. Similar plots representing the results from the noisy T-Maze task are shown in Figure 4.11. In both cases, the top row of plots represented the task for which the agent had to take the action corresponding to the Q-value 0 (i.e. the blue lines) at the T-junction, while the bottom rows of

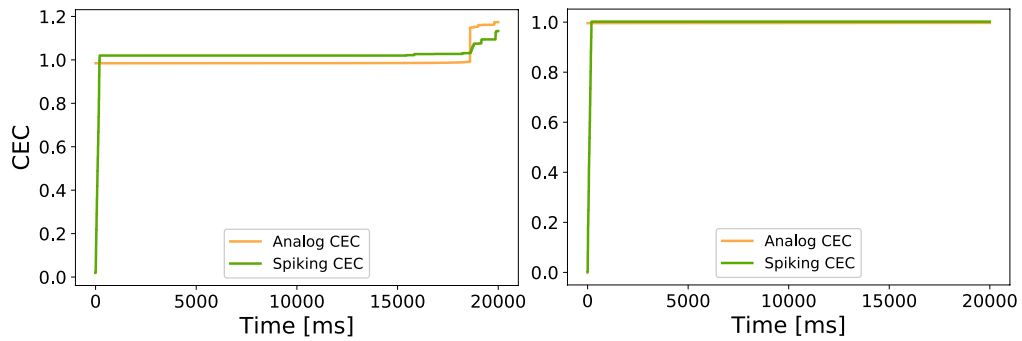

Figure 4.9: *Update of the CECs during the noiseless (left panel) and noisy (right panel) Sequence Prediction tasks.*

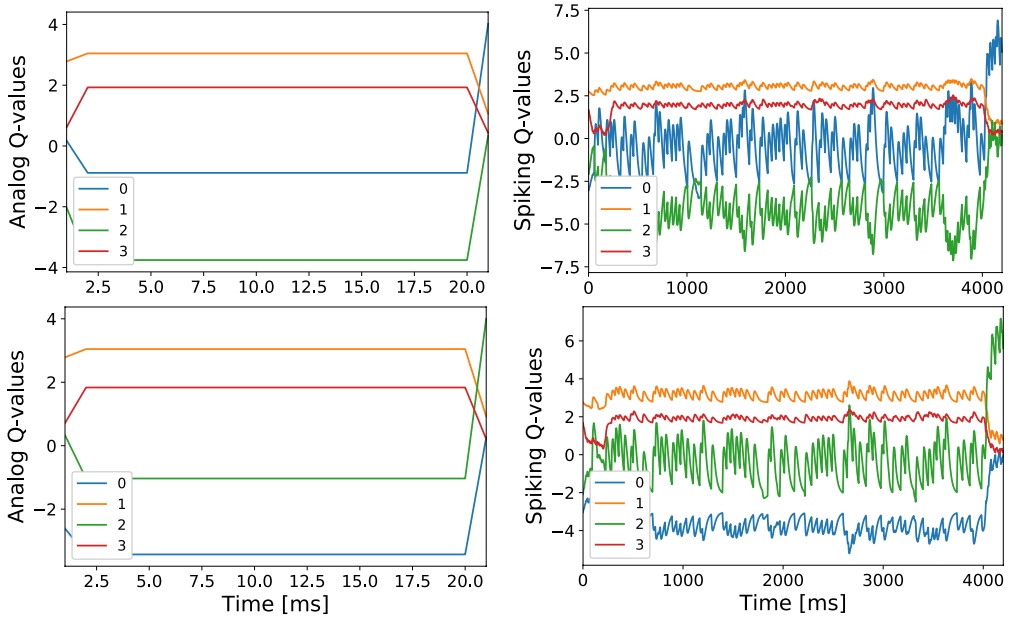

Figure 4.10: *Q-values of the analog (left) and spiking (right) network for the noiseless T-Maze task. Top row: task for which the action 0 (i.e. "go up") needed to be chosen at the T-junction; bottom row: task for which the action 2 (i.e. "go down") needed to be taken instead.*

these two groups of plots show the evolution over time of the Q-values when the input corresponding to the action 2 needed to be chosen at the end of the maze by the agent. In Figure 4.12 the evolution over time of the CECs present in the networks

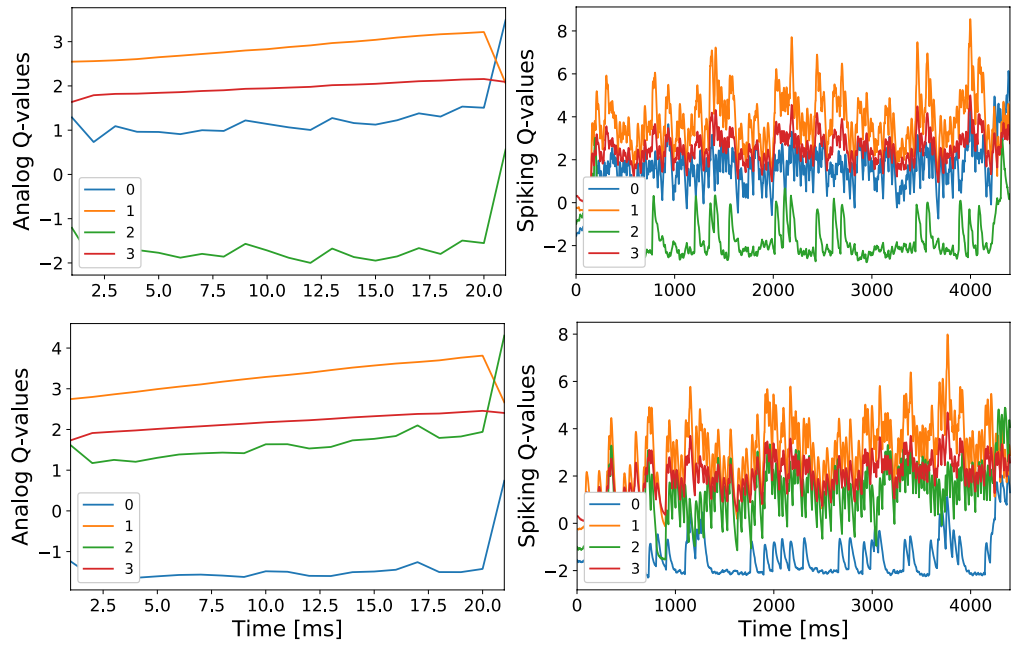

Figure 4.11: *Q-values of the analog (left) and spiking (right) network for the noisy T-Maze task. Top row: task for which the action 0 (i.e. "go up") needed to be chosen at the T-junction; bottom row: task for which the action 2 (i.e. "go down") needed to be taken instead.*

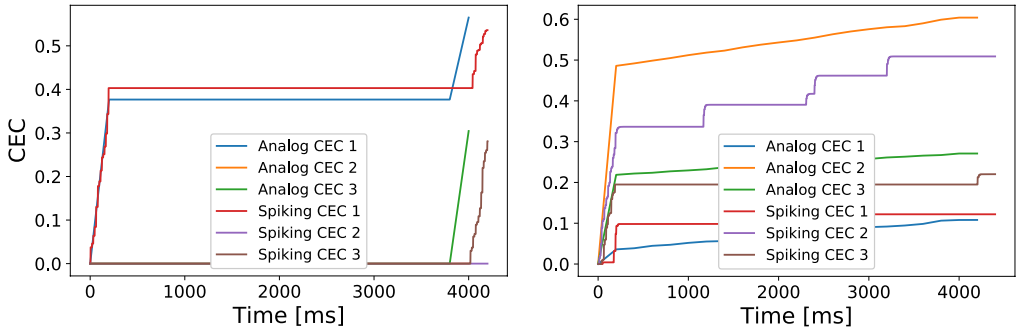

Figure 4.12: *Update of the CEC during the noiseless (left) and the noisy (right) T-Maze tasks.*

for the noiseless (left panel) and noisy (right panel) T-Maze tasks are shown. It can be noticed that the CEC of the spiking LSTMs reach different values compared to their analog counterparts. This is probably due to the increased network and task complexity.

In conclusion, the overall results for the Sequence Prediction and the T-Maze tasks are summarized in Table 4.1. All of the networks that were successfully trained for the noise-free and noisy Sequence Prediction tasks could be converted into spiking networks. Indeed, for the 19 successful networks, after presenting either x or y as the first symbol of the sequence, the average error over the last 200ms was always below the chosen threshold of 0.25. In the noiseless task the number of trials needed to reach the convergence criterion were, on average, even lower than the one reported in [9]. Similar results were obtained for the T-Maze task: all the networks were successful after the conversion in both the noise-free and noisy conditions. Finally, it must be emphasized that the firing rates reported show reasonable (low) values compatible with the values (found in the literature) recorded from real neurons.

Task	Orig. C.I. (%)	AAN C.I. (%)	ASN (%)	N_{spikes} (Hz)
noiseless Seq. Prediction	5040 (100)	2574 (95)	95	7129(71)
noisy Seq. Prediction	5680 (100)	8732 (95)	95	7114(71)
noiseless T-Maze $N = 20$	$1M$ (100)	8990(100)	100	2844(25)
noisy T-Maze $N = 20$	$1.75M$ (100)	71531(80)	80	4261(38)

Table 4.1: *Summary of the results. The second column refers to the results obtained from the original experiments [9, 61], the others to the present study; C.I. stands for* convergence iterations*: for the Sequence Prediction tasks the number of iterations corresponds to the number of episodes, while for the T-Maze tasks it corresponds to the total number of steps. ASN accuracy (%), total number of spikes per task and firing rate (Hz) are also reported.*

Chapter 5

Discussion and outlook

In this chapter, the final considerations concerning the work done are made, together with some future perspectives inherent the research involving spiking LSTM networks.

Regarding the more biologically plausible analytical transfer function derived, it gave promising results in terms of low firing rate, but by testing it in the three-neuron architecture it showed that the problem of mismatching CECs persisted. Further studies can be done, in particular about the refractoriness of the neurons, in order to check whether by removing these sources of error a better superposition can be obtained. An idea could be to model theoretically the refractoriness so that only the stable part of the output signal could be extracted and a more accurate coding could be obtained. Another plausible reason found in literature for this error could be the unreliability of the synapses in transmitting signals in the same way over time[64].

Moreover, still concerning the analytical transfer function, it could be possible to derive a versatile transfer function in which the time constants appear, so that it would allow to describe the behavior of the spiking neurons for any set of parameter values. However, the implementation of the stochastic neuronal model in order to find a good conversion from analog to spiking neurons was prioritized.

The stochastic transfer function obtained by means of the fitting procedure described in the previous chapters, instead, was shown to be able to accurately characterize the behavior of the ASNs. In fact, not only the spiking CEC showed a very accurate update in parallel with the analog one when implemented in a simple architecture, but also when the spiking LSTM was implemented in complicated tasks, it gave very good results.

Hence, thanks to the stochastic neuronal model, a very accurate mapping from analog to spiking neurons was found. Other studies had already proposed the same solution, the difference is that in the present work a gated memory unit was included as well, leading to very promising results in learning and solving tasks. However, an interesting outcome could be derived from the improvement of training procedures performed on SNNs. There are already, in fact, many ongoing research studies on this topic, but a good solution does not seem to exist, yet. This could help in the testing of spiking LSTM networks, since the errors coming from the not-flawless superposition between AANs and ASNs would be avoided.

It needs to be underlined in any case that if the goal is merely the efficiency, there is no reason to send spikes internally from the input cell and the CEC: the

conversion into spikes is useful if there are more connections than neurons (i.e. not the present case), because it leads to fewer matrix multiplications. This work has to be seen, instead, as an inquiry into the future development of fully functioning and energy-efficient spiking networks, and as a small step towards their deployment in advanced and complex research.

The fact that this function is more biologically plausible represents a strong point of this study, since it opens the path towards a field of research for computational efficiency optimization of asynchronous memory systems. The only exception, in the biological plausibility of the architecture presented here, is the way multiplicative gating mechanisms are included. In these regards, additive methods were recently proven to work[65], and they could be implemented in this context in future research.

Furthermore, many additional interesting aspects could be exploited concerning the structure of the LSTM unit. In fact, the LSTMs used here are not too complex, since they involve only one gate (with the CEC input regulated by modulating the decaying factor of the CEC) and no recurrent connections were implemented, but this architecture comprises of everything needed for the tasks studied. However, with the implementation of the forget gate and the recurrent weights, the range of applicability for the spiking LSTM networks could be greatly expanded. In many tasks, in fact, it is necessary to reset the CEC to discard long term dependencies, and the forget gate would operate in this sense. In addition, other studies have shown that a combination of input and forget gates can outperform LSTM on a variety of tasks while reducing the LSTM complexity[66, 67, 10]. For this reason, the implementation of the forget gate should be the next step in the research involving spiking LSTM networks.

Acknowledgements

After an intensive period of four months, I would like to reflect on the people who have supported me during the course of this project.

First, I would like to thank my thesis supervisor, Dr. Sander Bohte, for his support, patience, knowledge, and guidance, and his time spent correcting the italicisms spread all over my thesis.

Special thanks go to my daily supervisor, Dr. Davide Zambrano ("The supreme boss", since the night he fixed a bug in the T-Maze code), not only for being an immense font of knowledge I could tap into–literally–anytime I needed to, but in particular for being an all-around life coach and a dear friend (despite the sassiness). I cannot express how grateful I am for the time we have spent together.

My personal team of proof-readers and advisers, i.e. my scattered-all-over-Europe friends, need to be individually mentioned: Matteo, my living landmark, for his selfless time and care, Frida, for giving me bits of happiness even in the darkest times with the most amusing and the wisest Snap-conversations ever, Elia, for being my best (low-maintenance) friend and for all the hilarious chats we always have together, Hamzah, for the endless nights spent talking and laughing, and Giulia, for being the most caring, generous and the toughest friend I have. Thank you, guys, from the bottom of my heart, for being part of my life.

Finally, I thank my parents for the life-long support and for having faith in me through all the twists and turns, my sister for always putting up with me, and all the other friends and relatives who have accompanied me throughout the years with nice memories.

Bibliography

[1] F. Rieke, D. Warland, R. d. R. van Steveninck, and W. Bialek, "Spikes: Exploring the Neural Code."

[2] M. DeWeese and W. Bialek, "Information flow in sensory neurons," *Il Nuovo Cimento D*, vol. 17, no. 7, pp. 733–741, 1995.

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

[4] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*. Martin Hagan, 2014.

[5] B. DasGupta and G. Schnitger, "The power of approximating: a comparison of activation functions," in *Advances in neural information processing systems*, pp. 615–622, 1993.

[6] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural networks*, vol. 1, no. 1, pp. 17–61, 1988.

[7] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," *Diploma, Technische Universität München*, vol. 91, 1991.

[8] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

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

[10] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems*, 2017.

[11] D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells," *Josa a*, vol. 4, no. 12, pp. 2379–2394, 1987.

[12] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A Survey of Neuromorphic Computing and Neural Networks in Hardware," *arXiv preprint arXiv:1705.06963*, 2017.

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

[14] W. Gerstner and W. Kistler, "Spiking Neuron Models Cambridge University Press," 2002.

[15] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[16] E. D. Adrian, "The impulses produced by sensory nerve endings," *The Journal of physiology*, vol. 61, no. 1, pp. 49–72, 1926.

[17] P. Dayan, L. Abbott, and L. Abbott, "Theoretical neuroscience: computational and mathematical modeling of neural systems," *Philosophical Psychology*, pp. 563–577, 2001.

[18] A. A. Ezhov and D. Ventura, "Quantum neural networks," in *Future directions for intelligent systems and information sciences*, pp. 213–235, Springer, 2000.

[19] A. Kapoor, N. Wiebe, and K. Svore, "Quantum perceptron models," in *Advances in Neural Information Processing Systems*, pp. 3999–4007, 2016.

[20] N. Kouda, N. Matsui, and H. Nishimura, "Image compression by layered quantum neural networks," *Neural Processing Letters*, vol. 16, no. 1, pp. 67–80, 2002.

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

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

[23] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[24] L. É. Lapicque, *L'excitabilité en fonction du temps: La chronaxie, sa signification et sa mesure*. Les presses universitaires de France, 1926.

[25] R. B. Stein, "The information capacity of nerve cells using a frequency code," *Biophysical journal*, vol. 7, no. 6, pp. 797–826, 1967.

[26] B. W. Knight, "Dynamics of encoding in a population of neurons," *The Journal of general physiology*, vol. 59, no. 6, pp. 734–766, 1972.

[27] W. R. Softky and C. Koch, "The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs," *Journal of Neuroscience*, vol. 13, no. 1, pp. 334–350, 1993.

[28] W. Gerstner, R. Ritz, and J. L. Van Hemmen, "Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns," *Biological cybernetics*, vol. 69, no. 5-6, pp. 503–515, 1993.

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

[30] C. Pozzorini, R. Naud, S. Mensi, and W. Gerstner, "Temporal whitening by power-law adaptation in neocortical neurons," *Nature neuroscience*, vol. 16, no. 7, pp. 942–948, 2013.

[31] U. Sivarajah, M. M. Kamal, Z. Irani, and V. Weerakkody, "Critical analysis of big data challenges and analytical methods," *Journal of Business Research*, vol. 70, pp. 263–286, 2017.

[32] Y. Wang, T. Tang, B. Li, L. Xia, and H. Yang, "Energy efficient spiking neural network design with RRAM devices," in *Emerging Technology and Architecture for Big-data Analytics*, pp. 245–259, Springer, 2017.

[33] H. Zaidi, C. Labbé, and C. Morel, "Implementation of an environment for Monte Carlo simulation of fully 3-D positron tomography on a high-performance parallel platform," *Parallel Computing*, vol. 24, no. 9, pp. 1523–1536, 1998.

[34] J. Giersch, A. Weidemann, and G. Anton, "ROSI—An object-oriented and parallel-computing Monte Carlo simulation for X-ray imaging," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 509, no. 1, pp. 151–156, 2003.

[35] S.-J. Tu, C. C. Shaw, and L. Chen, "Noise simulation in cone beam CT imaging with parallel computing," *Physics in medicine and biology*, vol. 51, no. 5, p. 1283, 2006.

[36] M. Doulgerakis, A. Eggebrecht, S. Wojtkiewicz, J. Culver, and H. Dehghani, "Toward real-time diffuse optical tomography: accelerating light propagation modeling employing parallel computing on GPU and CPU," *Journal of biomedical optics*, vol. 22, no. 12, p. 125001, 2017.

[37] M. Garofalo, A. Botta, and G. Ventre, "Astrophysics and Big Data: Challenges, Methods, and Tools," *Proceedings of the International Astronomical Union*, vol. 12, no. S325, pp. 345–348, 2016.

[38] J. VanderPlas, A. J. Connolly, Ž. Ivezić, and A. Gray, "Introduction to astroML: Machine learning for astrophysics," in *Intelligent Data Understanding (CIDU), 2012 Conference on*, pp. 47–54, IEEE, 2012.

[39] K. Schawinski, C. Zhang, H. Zhang, L. Fowler, and G. K. Santhanam, "Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit," *Monthly Notices of the Royal Astronomical Society: Letters*, vol. 467, no. 1, pp. L110–L114, 2017.

[40] T. LHC Study Group *et al.*, "The large hadron collider, conceptual design," tech. rep., CERN/AC/95-05 (LHC) Geneva, 1995.

[41] D. Carmi, A. Falkowski, E. Kuflik, T. Volansky, and J. Zupan, "Higgs after the discovery: a status report," *Journal of High Energy Physics*, vol. 2012, no. 10, p. 196, 2012.

[42] J. Becla, A. Hanushevsky, S. Nikolaev, G. Abdulla, A. Szalay, M. Nieto-Santisteban, A. Thakar, and J. Gray, "Designing a multi-petabyte database for LSST," *arXiv preprint cs/0604112*, 2006.

[43] H. R. Powell, "X-ray data processing," *Bioscience reports*, vol. 37, no. 5, p. BSR20170227, 2017.

[44] D. Reid, A. J. Hussain, and H. Tawfik, "Financial time series prediction using spiking neural networks," *PloS one*, vol. 9, no. 8, p. e103656, 2014.

[45] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," *Communications of the ACM*, vol. 57, no. 6, pp. 13–15, 2014.

[46] W. Zhao, G. Agnus, V. Derycke, A. Filoramo, J. Bourgoin, and C. Gamrat, "Nanotube devices based crossbar architecture: toward neuromorphic computing," *Nanotechnology*, vol. 21, no. 17, p. 175202, 2010.

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

[48] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, p. 201604850, 2016.

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

[50] A. Shrestha, K. Ahmed, Y. Wang, D. P. Widemann, A. T. Moody, B. C. Van Essen, and Q. Qiu, "A Spike-Based Long Short-Term Memory on a Neurosynaptic Processor,"

[51] S. Bohte, "Efficient Spike-Coding with Multiplicative Adaptation in a Spike Response Model," in *NIPS 25*, pp. 1844–1852, 2012.

[52] R. Jolivet, A. Rauch, H.-R. Lüscher, and W. Gerstner, "Predicting spike timing of neocortical pyramidal neurons by simple threshold models," *Journal of computational neuroscience*, vol. 21, no. 1, pp. 35–49, 2006.

[53] R. Naud, "The Dynamics of Adapting Neurons," 2011.

[54] D. Zambrano and S. Bohte, "Fast and Efficient Asynchronous Neural Computation with Adapting Spiking Neural Networks," *arXiv preprint arXiv:1609.02053*, 2016.

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

[56] D. Zambrano, R. Nusselder, H. S. Scholte, and S. Bohte, "Efficient Computation in Adaptive Artificial Spiking Neural Networks," *arXiv preprint arXiv:1710.04838*, 2017.

[57] R. Nusselder, "Master's Thesis: Spike-based Long Short-Term Memory networks," 2017.

[58] S. Denève and C. K. Machens, "Efficient codes and balanced networks," *Nature neuroscience*, vol. 19, no. 3, pp. 375–382, 2016.

[59] I. R. Fiete, "Learning and coding in biological neural networks," *Diss. Harvard University Cambridge, Massachusetts*, 2003.

[60] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[61] B. Bakker, "Reinforcement Learning with Long Short-Term Memory," in *NIPS 14*, pp. 1475–1482, 2002.

[62] M. Harmon and L. Baird III, "Multi-player residual advantage learning with general function approximation," *Wright Laboratory*, pp. 45433–7308, 1996.

[63] M. Wiering and J. Schmidhuber, "HQ-learning," *Adaptive Behavior*, vol. 6, no. 2, pp. 219–246, 1997.

[64] A. Zador, "Impact of synaptic unreliability on the information transmitted by spiking neurons," *Journal of neurophysiology*, vol. 79, no. 3, pp. 1219–1229, 1998.

[65] R. Costa, I. A. M. Assael, B. Shillingford, N. de Freitas, and T. Vogels, "Cortical microcircuits as gated-recurrent neural networks," in *Advances in Neural Information Processing Systems*, pp. 271–282, 2017.

[66] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[67] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

Appendix

Biophysical models for the neurons

Here, some details of the biophysical models for the neuron are presented, in particular for the Hodgkin-Huxley and the FitzHugh-Nagumo models.

Hodgkin-Huxley

From basic thermodynamics it results that a voltage Δu, i.e. the Nernst potential, is generated by a difference in ion density between two regions described (at the equilibrium) by the formula:

$$\Delta u = \frac{kT}{q} \ln \frac{n_2}{n_1}\,, \tag{1}$$

with n_1 and n_2 being the ion concentrations of two different regions, k the Boltzmann constant, and T the temperature. At the equilibrium, differences in ionic concentrations cause a Nernst potential at the level of the neuronal membrane of about -65 mV (i.e. the previously-mentioned resting potential). Thus, the cell membrane can be seen as a capacitor and if a current $I(t)$ is applied it can be split into a component I_C charging the capacitor and other parts I_k passing through the k ion channels:

$$I(t) = I_C(t) + \sum_k I_k(t)\,. \tag{2}$$

The charging current can be written as $I_C = C\frac{\mathrm{d}u}{\mathrm{d}t}$ and the current flowing through the ion channels is given by their conductance $g_k = 1/R = u - E_k$, with E_k being the voltage for a specific channel. However, the conductance is generally time- and voltage-dependent, and the channels behavior is not easily described, e.g. it can happen that they are also blocked. Here lies the significance of Hodgkin and Huxley's study: by introducing some gating variables reflecting the probability for the channels to be open, they managed to find a description for the resistance of each ion channel as a function of time and voltage. The evolution of each gating variable x is given by the following differential equation:

$$\dot{x} = -\frac{1}{\tau_x(u)}[x - x_0(u)]\,, \tag{3}$$

or, by means of the variable transformations $x_0(u) = \alpha_x(u)/[\alpha_x(u) + \beta_x(u)]$ and $\tau_x(u) = [\alpha_x(u) + \beta_x(u)]^{-1}$, the channel gating can be also expressed in terms of the voltage-dependent transition rates α and β:

$$\dot{x} = \alpha_x(u)(1 - x) - \beta_x(u)x\,. \tag{4}$$

In principle, a complete biophysical model for the neuron can be built if both the channels present and their respective parameters are known. Originally, Hodgkin and Huxley included in their theory only two ionic channels, i.e. sodium and potassium, and a leakage channel. Under this assumption, the resulting ion currents are expressed as

$$\sum_k I_k = g_{\mathrm{Na}} m^3 h(u - E_{\mathrm{Na}}) + g_{\mathrm{K}} n^4 (u - E_{\mathrm{K}}) + g_{\mathrm{L}}(u - E_{\mathrm{L}})\,, \tag{5}$$

with the gating variables m and h controlling the evolution of the sodium channels, and the gating variable n controls the potassium channels. The subscript L refers to unspecific leakage channels, a voltage-independent type of resistor. Substituting the right-hand terms in Equation 2 with the detailed formulations, together with the evolution of the three gating parameters m, n, and h given by Equation 3, gives the four nonlinear differential equation system foundation of the Hodgkin-Huxley model, presented in the Introduction (Equations 1.1).

In particular, given that m_0 and n_0 increase with u, while the opposite applies to h_0, the generation of action potentials can be described as follows: if the membrane voltage increases because of an external input, the conductance of the sodium channels increases, leading to sodium ions flowing into the cell and raising the membrane potential even more. Depending on the extent of this rise, the potential can reach a–positive–critical point leading to action potential initiation. Thus, a threshold-type behavior characterizes the neuron, and the spike initiation depends purely on the charge delivered or, equivalently, on the value of the membrane potential immediately after an input current is received. However, it must be mentioned that the threshold is not well-defined mathematically, and the description used depends on the characterization of the input current.

FitzHugh-Nagumo

The–general–approach for going from four down to two differential equations comprises of applying the quasi steady-state approximation to the equation involving the gating variable m, since it varies quickly respect both to n and h and to a passive membrane. Thus, thanks to the separation of time scales between fast and slow variables, m can be treated as an instantaneous variable. Moreover, concerning the other two gating variables n and h, it was noticed that their time constants evolve in a similar manner over the voltage u and the same applies to the graphs of $n_0(u)$ and $1 - h_0(u)$. So, as a further approximation, a single variable w can be used instead of n and $1 - h$ (or, more generally, $b - h \approx an$, with a and b constants). In this way, $h = w - b$ and $n = w/a$ and, with $m = m_0(u)$, the first of the four differential equations at the heart of the Hodgkin-Huxley model becomes:

$$C\frac{\mathrm{d}u}{\mathrm{d}t} = -g_{\mathrm{Na}}[m_0(u)]^3(b - w)(u - E_{\mathrm{Na}}) - g_{\mathrm{K}}\Big(\frac{w}{a}\Big)^4 - g_{\mathrm{L}}(u - E_L\mathrm{L}) + I\,, \tag{6}$$

and, since the equation for the evolution of m disappeared, it results in the system of equation presented in the Introduction (Equations 1.2).

The previous model

In this appendix the mathematical steps followed in [57] in order to derive an analytical transfer function describing the output of the deterministic ASNs as described in Chapter 2 are presented.

In that thesis, the transfer function with $\tau_\eta = \tau_\gamma$ was obtained by following a reasoning similar to the one presented in Section 2.2.1 until Equation 2.19. From there, the substitution $\tau_\gamma = \tau_\eta$ was applied, obtaining:

$$(2S+\vartheta_0)e^{-\frac{t_s}{\tau_\eta}} - [2S(m_f+1)+\vartheta_0]e^{-2\frac{t_s}{\tau_\eta}} + [2S(m_f+1)-\vartheta_0]e^{-\frac{t_s}{\tau_\eta}} = 2S-\vartheta_0\,,$$

$$e^{-\frac{t_s}{\tau_\eta}} =: x \quad \Longrightarrow \quad [2S(m_f+1)+\vartheta_0]x^2 - [2S(m_f+2)]x + 2S - \vartheta_0 = 0\,,$$

which gives the solution:

$$e^{-\frac{t_s}{\tau_\eta}} = \frac{2S-\vartheta_0}{2S(m_f+1)+\vartheta_0}\,.$$

By means of Equation 2.15, then, the transfer function is obtained:

$$f(S) = \frac{h\cdot\tau_\eta}{t_s} = \frac{-h}{\ln\left(\frac{2S-\vartheta_0}{2S(m_f+1)+\vartheta_0}\right)}\,.$$

In addition, a schematic depiction of the LSTM unit used when this transfer function was tested in the same tasks as here is given in Figure 1 (the σ in AANσ and ASNσ refers to the just-found sigmoid-like transfer function). For more details, see the thesis [57].

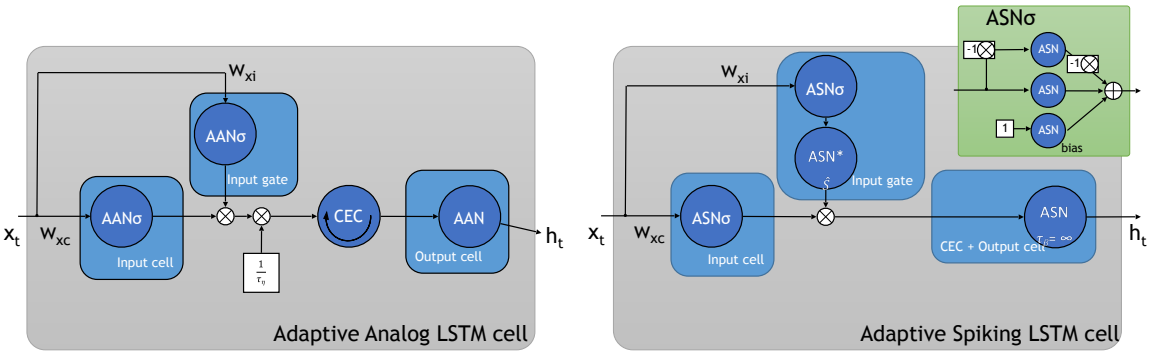

Figure 1: *Depiction of the LSTM architectures used originally. For more details, see [57].*